Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

Aumento de resolución de la nube de puntos generada por un LiDAR mediante técnicas de Deep Learning

Autor: Bruno Nicolás Valdebenito

Tutor: Luis Miguel Bergasa Pascual

2020

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Trabajo Fin de Grado

Aumento de resolución de la nube de puntos generada por un LiDAR mediante técnicas de Deep Learning

Autor: Bruno Nicolás Valdebenito

Director: Luis Miguel Bergasa Pascual

Tribunal:

Presidente: Enrique Santiso Gómez

Vocal 1: Alejandro Martínez Arribas

Vocal 2: Luis Miguel Bergasa Pascual

Calificación: .....

Fecha: .....

A todos los que me acompañaron
y apoyaron durante esta etapa

Agradecimientos

Esta es, de alguna manera figurativa, la conclusión de una etapa muy importante en mi vida y el comienzo de otra nueva. En este trabajo están plasmados sueños que un día decidí perseguir, y hoy se cumplen.

Le doy gracias a Dios por estar donde estoy. Empezar esta carrera hace cuatro años, en un país nuevo, en una cultura diferente, lejos de la familia y amigos que un día me rodearon, no fue fácil.

Agradezco primeramente a los que estuvieron siempre, a los que siempre confiaron en mí, a los que me apoyaron en todo, e hicieron hasta lo imposible para ayudarme a seguir adelante. Me animaron con cada gesto y me motivaron para seguir adelante y no rendirme, me aguantaron en cada momento de estrés y celebraron conmigo cada logro. Gracias Pa y gracias Ma, gracias a ustedes soy quien soy, y ustedes son los principales a quienes quiero dedicar este logro. Junto con ustedes, también agraezco a mis hermanos, Mica y Maxi, estuvieron siempre y me aguantaron en todo. Son la mejor familia que existe.

Los amo.

Gracias a mi familia y amigos en Argentina, que siempre me apoyaron en todo lo que pudieron, sin importar la distancia, estuvieron para animarme, apoyarme, ayudarme en lo económico, pero también y más importante en todo lo demás, en apoyarme, en animarme, en aconsejarme, en creer en mi, en quererme desde lejos y demostrarme su cariño a pesar de estar lejos

Gracias a mis amigos, a todos los que estuvieron conmigo, con quienes pude despejarme, pude relajarme, descargarme y supieron aguantarme, aunque sea bien cabezón. Gracias a esas personas que me acompañaron en llamadas nocturnas hasta altas horas de la madrugada mientras estudiaba, a aquellas que en momentos de estrés y bajón supieron sacarme risas.

Gracias a mis compañeros, con los que compartimos este viaje, los panas con los que compartimos estos cuatro años, todos los días de la semana. Enfrentamos cada exámen juntos, nos acompañamos los unos a los otros siempre. Estoy agradecido con ellos me enseñaron a dar el extra. Compartir estos cuatro años de mi vida con ustedes fue un auténtico placer, gracias a todos mis panas.

Por último agradecer a Luis Miguel Bergasa, y a todo el equipo de Robesafe, por permitirme realizar este trabajo con ellos, poder aprender y desarrollarme en areas nuevas, y ayudarme a crecer en el ámbito profesional.

Autor

Bruno Valdebenito

Estoy aquí porque Dios tuvo
planes conmigo, gracias a ustedes
y a los míos soy agradecido.
Tumbarme a mí mis enemigos no
han podido, yo estoy bendecido...

Resumen

Para la utilización de técnicas de detección y seguimiento de múltiples objetos en una escena urbana basados en LiDAR con un sensor de bajo coste, se estudiará en este Trabajo de Fin de Grado la conversión de datos obtenidos de un sensor LiDAR de 16 haces a datos de un sensor LiDAR de 64 haces, haciendo uso de algoritmos de Deep Learning.

El fin de este sistema a diseñar es aplicarlo en el área de percepción de vehículos autónomos, combinándolos con sistemas de detección, también basados en algoritmos de Deep Learning, con el objetivo de lograr un aumento de resolución en los datos de entrada en la detección, obteniendo de esta forma mejores resultados en la detección de objetos, al usar modelos entrenados con LiDARs de 64 haces. Logrando un rendimiento similar al de un sensor LiDAR de 64 haces, pero con uno de 16 haces, reduciendo costes en el hardware.

Particularmente se trabajará en el proyecto Tech4AgeCar, del grupo de investigación Robesafe [1], donde se parte de un coche ya equipado con varios sensores, entre ellos un LiDAR de 16 haces, y un sistema ya capaz de conducir automáticamente el vehículo por las calles del campus externo de la Universidad de Alcalá. El equipo de Robesafe ya tiene desarrollado también un entorno para la simulación de este sistema de manera virtual en el simulador CARLA, con la capacidad de utilizar los mismos sensores y realizar ensayos previos a la prueba en el entorno real.

Keywords: Conducción Inteligente, Robótica, sensor LiDAR, Percepción, Deep Learning.

Abstract

For the use of multi-object tracking and detection techniques in a LiDAR-based urban scene with a low cost sensor, this final degree project will consider converting data from a 16 channels LiDAR sensor to data from 64 channels LiDAR sensor, using Deep Learning algorithms.

The purpose of the developed system is to apply it in the area of perception of autonomous vehicles, combining them with detection systems, also based on Deep Learning algorithms, with the aim of increasing the resolution of the input data in the detection, and in addition to obtaining better results in the object detection, when using trained models with 64 channels LiDAR sensor. Achieving a similar performance to that of a 64 channels LiDAR sensor, but using a 16 channel sensor, reducing hardware costs.

Particulary it work will be done on the Tech4AgeCar project, from the Robesafe research group [1], where you will start from a car already equipped with several sensors including a 16 channels LiDAR, and a system already capable of automatically driving the vehicle through the streets of the external campus of the University of Alcalá. The Robesafe team has already developed an environment for simulation of this system in a virtual world in CARLA simulator, with the ability to use the same sensors and perform before test it in the real world.

Keywords: Smart Driving, Robotics, LiDAR sensor, Perception, Deep Learning.

Resumen extendido

Para minorar el problema del coste económico que tiene un sensor LiDAR de 64 haces y poder hacer uso de toda la información de estos sensores, se crea un sistema basado en algoritmos de Deep Learning, el cual a partir de nubes de puntos de un sensor LiDAR de 16 haces es capaz de interpolar nubes de puntos de 64 haces. De esta forma, se podrá utilizar un LiDAR de 16 haces como si fuese uno de 64 haces. La idea principal es aumentar la resolución de los datos adquiridos para luego utilizarlos para la parte de percepción del sistema de un coche autónomo.

El funcionamiento se basa en convertir la nube de puntos en una imagen, transformar dicha imagen de 16 haces a una de 64 mediante una red neuronal y realizar la conversión en sentido contrario, de imagen a nube de puntos. La imagen de entrada se procesará con una red neuronal tipo GAN (pix2pix Network) [2], diseñada para la traducción de imagen a imagen, algunos de los ejemplos que ofrecen son las conversiones de fotografías a pinturas, de dibujos de trazo a imágenes realistas de objetos, paisajes de verano a invierno, de día a noche, entre otros.

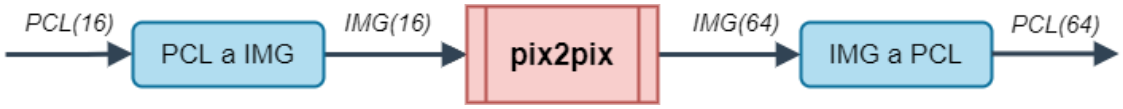

Figura 1: Diagrama de funcionamiento básico del sistema

En este caso se realizará la conversión de imágenes PGM de una nube de 16 haces a imágenes PGM de una nube de puntos de 64 haces. PGM son las siglas de Polar Grid Map, que es una especie de mapa de profundidad, pero en coordenadas polares. Los datos para entrenamiento se obtendrán a partir de una simulación en CARLA simulator. El sistema se integrará con ROS para poder implementarlo en la percepción, ya sea en simulación en CARLA o en el coche real del grupo de investigación Robesafe.

Finalmente, se comentan conclusiones obtenidas, tales como la precisión final, los resultados obtenidos al momento de utilizar las nubes de puntos generadas por el sistema para la percepción, las mejoras por hacer, etc.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

API	Application Programming Interfaces.
GAN	Generative Adversarial Network.
GPS	Global Positioning System.
IMU	Inertial Measurement Unit.
KNN	K-Nearest Neighbours.
LiDAR	Light Detection And Ranging.
PGM	Polar Grid Map.
PLY	Polygon File Format (formato).
PNG	Portable Network Graphics (formato).

Capítulo 1

Introducción

1.1 Contexto del trabajo

La conducción inteligente ha avanzado a pasos agigantados en los últimos años, es un área relativamente nueva en la investigación y desarrollo, y se está convirtiendo en una realidad. Las grandes empresas, tanto de la industria de los automóviles como también de la industria de la informática, fijaron sus objetivos en el futuro de los coches inteligentes y hoy en día solo es cuestión de tiempo para tener coches totalmente autónomos. Aparecen nuevas empresas dedicadas al sector de la inteligencia artificial, utilizando técnicas de Deep Learning y Machine Learning, siendo esta la base de la conducción inteligente.

El problema más grande que se plantea es que en una carretera hay vidas humanas en juego y no se pueden permitir errores al momento de percibir el entorno en el que se moverá el vehículo. Un entorno en constante cambio, con acciones completamente inesperadas, con muchos factores que cambian constantemente y suceptible a errores pueden ser causantes de graves accidentes, hasta letales.

Por esta razón, es esencial un sistema de percepción y predicción en los vehículos, para lo cual necesitan de sensores que sean capaces de permitir al sistema reconocer todos los posibles factores que pueden afectar en la conducción. Los sensores de los que son dotados los vehículos autónomos principalmente son:

- GPS
- IMU
- Cámara (RGB, de profundidad, etc)
- Radar
- LiDAR

Importante destacar para la percepción del entorno el conjunto de cámaras, las cuales proveen imágenes del entorno del vehículo, y los sensores LiDAR (Light Detection And Ranging), los cuales generan una nube de puntos en 3D de 360 grados, teniendo un mapa en tres dimensiones de todo el entorno del vehículo en un rango medianamente lejano. Estos dos sensores planteados son las dos opciones más utilizadas en el mundo de la percepción, y la combinación de ambas para el estudio del entorno es la estrategia más fiable a utilizar hoy en día, pero existe un gran dilema entre las compañías sobre si utilizar dispositivos baratos y pequeños como las cámaras, o dispositivos grandes, robustos y con gran resolución como los LiDARs, siendo muy dependiente esta elección de los algoritmos de detección que se utilicen.

Los beneficios de utilizar sensores LiDAR es principalmente la precisión que se tiene de cada punto alrededor del coche, que a diferencia de las cámaras RGB que solo dan información sobre la ubicación de los objetos en 2 dimensiones, las nubes de puntos también nos permiten conocer la profundidad con precisión en un rango alrededor de 100 metros, además de tomar datos de 360 grados, no de solo un área. Es necesario comentar que existen algoritmos para determinar profundidad con cámaras (que no sean exclusivamente cámaras de profundidad), por ejemplo, algoritmos que estudian la disparidad, los cuales utilizan el desenfoque, o utilizando 2 imágenes tomadas desde ángulos diferentes y a partir de la geometría conocer esta distancia.

En contra tenemos que los sensores LiDAR tienen un coste económico elevado, al ser aparatos de gran precisión son bastante costosos. Son sensores mucho más grandes que las cámaras, lo cual hace más compleja su implementación en la estructura de un coche normal. Existen varios tipos de LiDAR, entre ellos diferenciados por su angulo de detección, que pueden ser direccionales o de 360 grados, y estos últimos a su vez por la cantidad de haces que poseen, los más comunes son de 16, 32, 64 y 128 haces, siendo estos últimos los más caros. La mayoría de algoritmos de detección están diseñados para trabajar con un LiDAR de 64 haces, el cual tiene mayor precisión y puede hacer predicciones más exactas. Al momento de estudiar datos, o utilizar nubes de puntos ya etiquetadas y clasificadas para hacer entrenamiento, las bases de datos disponibles son la mayoría de LiDARs de 64 haces.

La solución a este problema planteado la podemos encontrar en el Deep Learning, no solo utilizándolo dentro del algoritmo de detección de objetos, sino también en la toma de datos. Existen pocos proyectos publicados que intenten abordar este problema con Deep Learning, uno de ellos y del cual se utilizaron algunas ideas como base para la investigación fue la publicación "LiDAR Sensor modeling and Data augmentation with GANs for Autonomous driving"[8], donde se plantea utilizar redes tipo GAN para data augmentation.

1.2 Objetivos del proyecto

El objetivo principal es desarrollar un sistema que interpole los datos de un sensor LiDAR de 16 haces a datos de un sensor LiDAR de 64 haces, a partir de algoritmos de Deep Learning, que sea capaz de aumentar la precisión de los datos iniciales, haciendo que los datos de salida se adapten a las características de un sensor de mayor precisión. A fin de utilizarlos con algoritmos de percepción, esperando tener mejores resultados que con el uso de los datos de entrada al sistema.

Los objetivos secundarios del TFG son los siguientes:

- Generar una base de datos a partir del simulador CARLA, con nubes de puntos en un mismo instante de dos sensores LiDAR, uno de 16 y otro de 64 haces, para poder hacer el entrenamiento de la red.
- Adaptar la base de datos (nubes de puntos) a una red neuronal pensada para trabajar con imágenes llamada pix2pix, de la rama de las GANs (Generative Adversarial Network), la cual modelará la construcción de nubes de puntos de un sensor de 64 haces a partir de nubes de puntos de un sensor de 16.
- Implementar el sistema completo con ROS, para poder ser utilizado en tiempo real en la percepción, tanto en simulación como en un coche real.

1.3 Plan de trabajo

Para abordar la investigación y comprobar la hipótesis de que es posible construir nubes de puntos de 64 haces a partir de la información proporcionada por nubes de puntos de 16, y obtener resultados similares a los esperados por un LiDAR de 64 haces, el plan de trabajo a realizar es el siguiente:

- Se estudian las posibles maneras de aumentar la resolución de una nube de puntos.
- Una vez resuelto la manera de realizar la conversión, se estudian las redes neuronales que podrían realizar la conversión.
- Se preparan pequeños datasets de prueba a partir de CARLA y KITTI para realizar ensayos de entrenamiento en las diferentes redes.
- Se elige la red y el dataset más apropiados para cada caso, se prepara un dataset más grande y se entrena la red.
- Se prepara la red con el procesamiento de los datos a la entrada y a la salida.
- Se monta el sistema en un Docker para facilitar su portabilidad.
- Se implementa con ROS para tomar datos, ya sea de una simulación en CARLA o de un sensor LiDAR real, y convertirlos para pasarlos al sistema de percepción.
- Se hacen pruebas y ensayos para obtener y estudiar los resultados.

Capítulo 2

Herramientas utilizadas

2.1 Introducción

Este capítulo comenta las herramientas software que se han utilizado para el desarrollo de este proyecto. Se explican brevemente para tener una idea clara sobre cada una de ellas y se comenta su funcionamiento dentro del desarrollo de este trabajo.

Estas herramientas software que se utilizarán son las siguientes:

- CARLA simulator
- KITTI dataset
- ROS
- Pix2pix Network
- Docker

2.2 CARLA Simulator [3]

Uno de los problemas más grandes al momento de hablar de Deep Learning y conducción autónoma es la necesidad de datos, una gran base de datos. Para entrenar un algoritmo de Deep Learning son necesarias grandes cantidades de datos, y el problema en la conducción inteligente es que obtener esos datos, tales como imágenes de un entorno urbano, mediciones inerciales de un trayecto, nubes de puntos, etc, significa que hay que poseer un coche dotado con los sensores necesarios para la obtención de dichos datos y este coche debe haber pasado un tiempo recibiendo esos datos.

Existen bases de datos de uso libre, como por ejemplo la base de datos de KITTI, la cual provee de datos ya etiquetados de distintos sensores, como cámaras de distintos tipos, un LiDAR de 64 haces, GPS, sensores de inercia, todos sincronizados. Estas pueden ser una buena opción para el desarrollo de sistemas como el que se está diseñando. El problema es que para este caso, esta base de datos no es ideal para trabajar, ya que los modelos de detección se han entrenado con datos de un sensor el cual no posee el vehículo, con el que su rendimiento no es óptimo.

Debido a este problema se ha optado por trabajar con el simulador CARLA, es un simulador de código abierto destinado a ser de ayuda para el desarrollo, entrenamiento y validación en sistemas de conducción

inteligente. CARLA proporciona un entorno virtual completamente editable, en el cual se pueden realizar pruebas con una gran cantidad de modelos de vehículos disponibles, con sensores los cuales pueden ser configurados de la manera que se desee. Se tiene el control completo de todos los actores que se quieran involucrar en el escenario, además de los factores que modelan el escenario mismo.

Figura 2.1: Entorno de simulación en CARLA

CARLA proporciona una potente API, para facilitar al usuario controlar todo de una manera sencilla durante la simulación. Además, proporciona programas de ejemplo, tanto en lenguaje Python como en C++, en los cuales existe la posibilidad de modificar todos los parámetros tales como el tipo de vehículo, su color, los sensores que posee, la ubicación de los sensores en el vehículo, las características de cada sensor, los demás vehículos y/o peatones que circularán por el mapa, y la posibilidad de obtener los datos tomados por todos los sensores.

Otro punto fuerte que aporta CARLA es un puente con ROS, para poder complementar ambas herramientas y facilitar la comunicación y poder implementarlo con otros sistemas al momento de realizar la validación.

La manera en la que se utilizará CARLA será para obtener dos conjuntos de datos. Serán nubes de puntos de sensores LiDAR, uno de 16 y otro de 64 haces, ubicados en la misma posición, para obtener una pareja de muestras tomadas en el mismo instante y en el mismo lugar, pero por dos LiDARs con diferente configuración. El entorno en que se tomarán las muestras es un mapa de una ciudad por defecto, con peatones, ciclistas y vehículos.

Figura 2.2: Sensor LiDAR en CARLA

2.3 KITTI dataset [4]

KITTI es una plataforma de conducción autónoma que permite utilizar abiertamente los datos obtenidos con diversos sensores, algunos de esos datos etiquetados para entrenamiento, y otros sin etiquetar para test y validación. Son datos tomados en situaciones reales de tráfico en escenas urbanas y se componen de los siguientes sensores:

- GPS/IMU
- LiDAR 64 haces
- 2 cámaras en escala de grises
- 2 cámaras a color
- 4 lentes vari-focales

Figura 2.3: Vehículo utilizado en la toma de datos para KITTI dataset

En este caso, lo principal par este TFG en este vehículo es el sensor LiDAR (Velodyne HDL-64E [9]), es un modelo de 64 haces, el cual aporta varias secuencias de tomas de datos, los cuales se procesarán para a partir de ellos obtener también nubes de puntos de 16 haces, y con todo este conjunto de datos trabajará la red que se desea entrenar.

2.4 Pix2pix Network[2]

Para la conversión de datos de un LiDAR de 16 haces a uno de 64 haces se hará uso de una red neuronal, la cual sea capaz de aumentar la resolución de unos datos de entrada a partir de una serie de parámetros con los cuales se desea obtener la salida de este algoritmo. Podía plantearse utilizar procedimientos geométricos de interpolación entre los puntos para generar este aumento de resolución de los datos, pero utilizando algoritmos de Deep Learning se llega a resultados que, por métodos meramente geométricos, no podrían obtenerse, llegando a ser estos últimos muy precisos y sobre todo útiles para la tarea a realizar.

Las redes GAN (Generative Adversarial Network) son un modelo de Deep Learning conformado por dos redes neuronales y utilizado en el aprendizaje no supervisado. Su funcionamiento es el siguiente, una de las redes es una red neuronal convolucional y se encarga de generar posibles datos de salida, esta es la red generadora, y la otra es una red neuronal deconvolucional la cual se encarga de evaluar la similitud entre los resultados de la red generadora y los datos de salida deseados, esta es la red discriminadora.

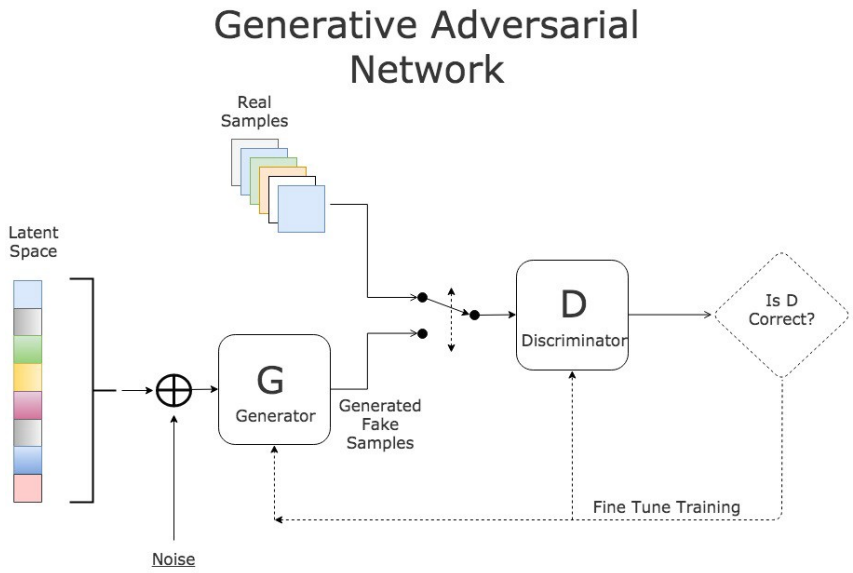

Figura 2.4: Diagrama de una red neuronal tipo GAN [7]

La red discriminadora se entrena con los datos deseados de salida, logrando que aprenda los parámetros que los definen. La red generativa intenta engañar a la red discriminadora, generando salidas lo más similares a los datos de salida deseados, ajustándose con la realimentación de la red discriminadora.

Su uso principal es con imágenes, en aplicaciones tales como colorear imágenes en blanco y negro, aumento de resolución, traducción imagen a imagen (image-to-image translate), y de este último podemos destacar casos como conversión imagen realista a pinturas, conversión de imágenes de caballos a cebras, dibujo (bordes) a foto realista, entre otros. Y esta última utilidad, la de traducción de imagen a imagen, es la principal característica de las GANs que hace que este tipo de redes sean útiles para la necesidad del proyecto.

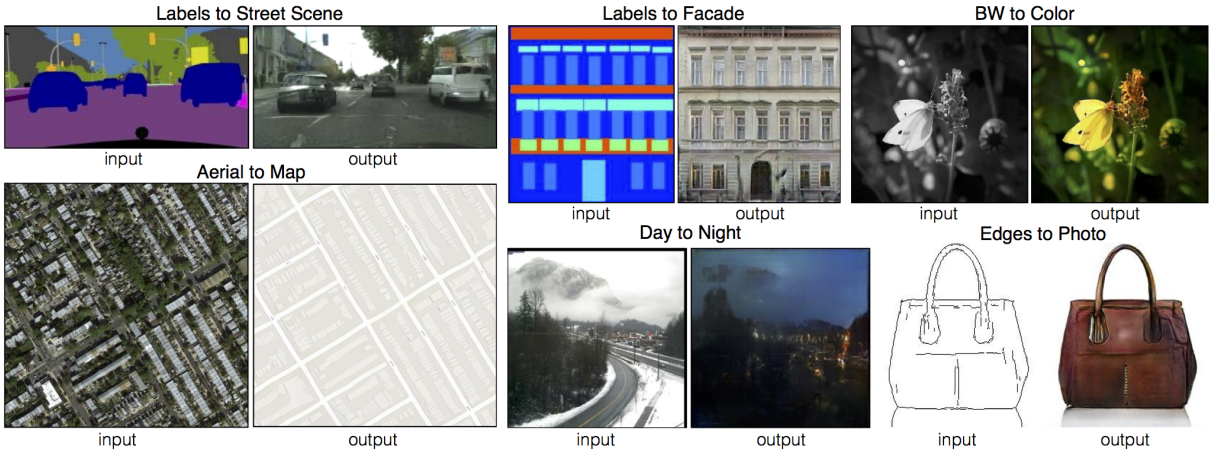

Figura 2.5: Ejemplos de funcionamiento de la red pix2pix

Se ha estudiado el funcionamiento de dos redes tipo GAN, la red cycleGAN y la pix2pix, pero con la

que se obtuvieron mejores resultados fue con la red pix2pix. Esta es una GAN condicional, es decir que el discriminador también conoce el dato de input además de la salida del generador. Su desempeño para la traducción imagen a imagen es muy bueno en términos cualitativos, se obtienen imágenes muy cercanas a las esperadas. El modelo que se utilizará está modelado en Python y permite la configuración de la red conforme a los parámetros que necesite el usuario, ya sea tamaño de la imagen, cantidad de canales (a color o escala de grises), elección de distintos tipos de las redes generadora y discriminadora, número de capas de cada una de ellas, tipo de preprocesamiento de los datos, entre otra más. También admite la configuración de parámetros de entrenamiento, como el tamaño del batch, la ganancia a la entrada, Learning Rate, número de iteraciones, entre otras.

La red pix2pix será la encargada de traducir los datos del LiDAR de 16 haces al de 64, esto se hará previamente convirtiendo las nubes de puntos obtenidas de CARLA a una PGM (Polar Grid Map), la cual es una representación en imagen de las nubes de puntos. Una vez transformados los datos, se preparará el dataset, asociando la entrada con la salida esperada correspondiente. Para lo anterior se hará el uso de Matlab. Una vez con el dataset preparado, se realizará el entrenamiento de la pix2pix, logrando así tener una red capaz de convertir imágenes obtenidas a partir de una nube de puntos de un sensor de 16 haces a imágenes que modelan la misma nube de puntos de la entrada, pero como si hubieran sido tomadas por el sensor de 64 haces.

Para llevar a cabo el entrenamiento debemos adaptar la red para que funcione como un programa al que le pasamos una nube de puntos y genera otra nube de puntos, similar a la de la entrada, pero de mayor densidad. En el sistema completo, anterior y posterior a este proceso, se debe realizar la conversión de nube de puntos a PGM y viceversa a la salida.

2.5 ROS [5]

Al momento de implementar el sistema de percepción completo se observa que está conformado por varias partes. Está formado por el sensor, el algoritmo de procesamiento de los datos y el algoritmo de detección de objetos, en términos generales. Se tienen partes hardware y partes software, y los módulos software son módulos que deberían ser capaces de ejecutarse en paralelo, pensando además en el resto de partes que forman el sistema de un coche autónomo. Es necesario un sistema sobre el que se implementen los módulos, que nos permita controlar la información y los sensores de una manera sencilla y con la posibilidad de ser flexible ante cambios o actualizaciones.

ROS (Robotic Operating System) es un framework pensado para el desarrollo de software para robots, el cual proporciona la funcionalidad de un sistema operativo, permitiendo la abstracción del hardware, el control de dispositivos de bajo nivel, el paso de mensajes entre procesos y el mantenimiento de los paquetes. ROS provee el sistema operativo y un conjunto de paquetes (ros-pkg), los cuales implementan las múltiples funcionalidades para el manejo de la información y el control hardware.

Con ROS el sistema será capaz de comunicarse y tener la capacidad de trabajar en tiempo real, además de poder conectarse con CARLA a través del ROS bridge que se proporciona y poder implementarse en el coche real.

2.6 Docker [6]

Docker es un software utilizado para crear contenedores portables y ligeros para las aplicaciones software, los cuales pueden ejecutarse en cualquier equipo con Docker instalado, siendo independiente del

sistema operativo que tenga el equipo. Dentro de este contenedor se tiene instalado las versiones de los diferentes recursos necesarios tales como softwares, paquetes, drivers, etc., que permiten que la aplicación desarrollada dentro de un Docker pueda ejecutarse en cualquier máquina.

Esta herramienta la utilizaremos para montar la red una vez entrenada, realizando las comunicaciones con ROS, teniendo un sistema completamente preparado para recibir nubes de puntos de 16 haces y devolver las nubes de puntos con aumento de resolución. En este Docker están configurados e instalados todos los paquetes necesarios para que se ejecute el sistema.

Capítulo 3

Desarrollo del proyecto

3.1 Introducción

En este capítulo se explican cada uno de los procesos para llevar a cabo la realización del sistema, así como también se explican en detalle los procedimientos que se llevaron a cabo para cada una de las tareas y pruebas que se realizaron para obtener la manera óptima de aumentar la resolución en las nubes de puntos.

También se explican con precisión la manera exacta en la que se utilizaron las herramientas anteriormente presentadas, y se muestra la evaluación de los resultados de los ensayos realizados antes de obtener el modelo final del sistema completo.

El orden que se sigue es un orden cronológico ideal, es decir que se explican las tareas desarrolladas secuencialmente, aunque debido a que este es un proyecto de investigación, cuando se ha llegado a resultados no deseados, se ha tenido que retroceder y rehacer algunas tareas, realizando cambios hasta poder comprobar la hipótesis inicial de poder realizar un aumento de resolución de nubes de puntos, aumentando el número de haces, las cuales serían obtenidas por un sensor LiDAR.

3.2 Preparación del dataset

En algoritmos de Deep Learning, los datos con los que se preparará el algoritmo son la pieza fundamental para cualquier aplicación en esta área, y para la aplicación que se busca desarrollar, el dataset necesario para el entrenamiento de la red neuronal pix2pix es peculiar. Como se comentó anteriormente son necesarias 2 muestras de nubes de puntos que hayan sido tomadas en el mismo punto y en el mismo instante por dos sensores LiDAR diferentes, uno de 16 haces y otro de 64.

Por esta razón se tienen 2 soluciones posibles, la primera es crear el dataset desde cero a partir de simulación y la segunda es modificar un dataset de 64 haces quitándole la información procedente de haces intermedios, para dejarle solo 16. En este proyecto se han abordado ambas soluciones para estudiar los mejores resultados.

3.2.1 Creación del dataset en CARLA simulator

Se utilizará el simulador CARLA, anteriormente mencionado, para la creación de los datos que permitirán entrenar a la red pix2pix. Para dicho procedimiento se ha creado un programa en Python, con la API

que proporciona CARLA, para la simulación de un escenario urbano con actores tales como coches, motocicletas, bicicletas y peatones. En dicho escenario tenemos un vehículo en piloto automático que circula por las calles y tiene 2 sensores LiDAR en la misma posición, pero un sensor es de 16 haces y el otro de 64 haces.

El programa está basado en un ejemplo que facilita CARLA a modo de tutorial. El procedimiento que sigue es, primero se crea el cliente que se conectará al mundo de CARLA, a continuación, se crea el vehículo que tomará los datos junto con los dos sensores LiDAR, se configuran cada uno de ellos y se unen al vehículo anteriormente creado. Los LiDARs se configuran para que tomen muestras cada 1 segundo y que se guarden en carpetas diferentes los datos de 16 haces y los de 64. Luego se crean el resto de actores que circularán en el escenario de forma automática y se programará el tiempo de simulación, que dependiendo de la cantidad de muestras que se deseen será diferente. Finalmente se matan todos los actores de la simulación.

Importante comentar que para el estudio de las redes se ha experimentado con múltiples configuraciones de los sensores LiDARs, realizando variaciones entre rango de profundidad, número de puntos de la nube (puntos por segundo), frecuencia del LiDAR, entre otras. Por lo cual se obtuvieron varias datasets para evaluar con cual configuración se llega a mejores resultados y con cual funciona mejor cada red.

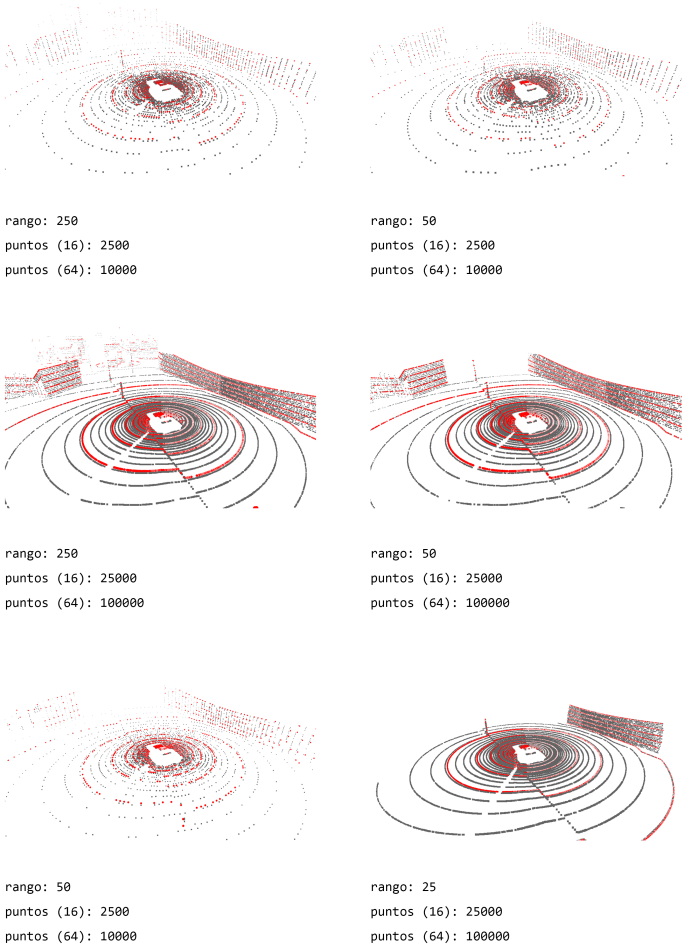

Figura 3.1: Diferentes parejas de nubes de puntos obtenidas

Como se puede observar en la figura 3.1, estas son las diferentes configuraciones con las cuales se tomaron datos en CARLA. Se puede ver la variación de puntos entre las diferentes muestras tomadas,

siendo la nube de puntos roja la correspondiente al sensor de 16 haces y la nube de puntos gris la correspondiente a la de 64 haces.

3.2.2 Creación del dataset a partir de KITTI dataset

Los datos que nos proporciona KITTI son nubes de puntos de un LiDAR de 64 haces, las cuales tienen aproximadamente 120.000 puntos y están codificadas en formato BIN. Estas nubes de puntos se modificarán en Matlab, creando un programa que primeramente realiza la lectura del fichero y luego separa en los diferentes haces que componen la nube de puntos por métodos geométricos. Las nubes se leen como matrices Nx3 donde N es el número de puntos que existen, no están separados por haces. Cuando se toman los datos del LiDAR sin procesar, los datos vienen codificados según el ángulo del haz con el que se tomó cada punto, pero una vez procesados y guardados, no hay manera de saber a cuál haz corresponde cada punto. Usando métodos geométricos, transformando la nube de puntos a coordenadas polares y dividiendo los puntos según el ángulo de elevación, se puede hacer una separación por haces bastante similar a la real, como se muestra en la figura 3.2.

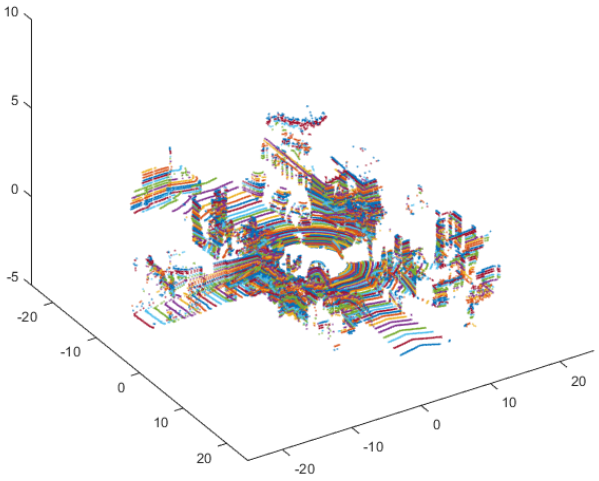

Figura 3.2: Separación de haces realizada por métodos geométricos

Cada color es un haz diferente, y como se puede ver, al ser una nube de puntos real, tomada en un entorno urbano real, se puede observar que la división de haces no es completamente correcta. Se puede observar que algunos de los haces obtenidos toman 2 haces reales y los agrupan como si fuese uno. Este es un pequeño error que se comete al momento de transformar la nube de 64 haces a 16 por estos métodos.

Una vez separados los 64 haces, se han generado nubes de puntos de 16 haces utilizando un rango de elevación similar al de un LiDAR con el que se han tomado los datos, el cual tiene un rango de 2 a -25 grados aproximadamente. De los 64 haces se tomaron desde el haz 2 hasta el 62, descartando la información de 3 haces intermedios (los haces utilizados son 2, 6, 10, 14, 18, ..., 58, 62). Se puede visualizar en la figura 3.3, siendo la nube en gris la nube de 64 haces, y la nube en color negro la de 16 haces.

También se generaron nubes de puntos de 16 haces utilizando solo la información más relevante, ya que los haces más bajos pertenecen a los que tienen un menor ángulo de elevación, y en la observación

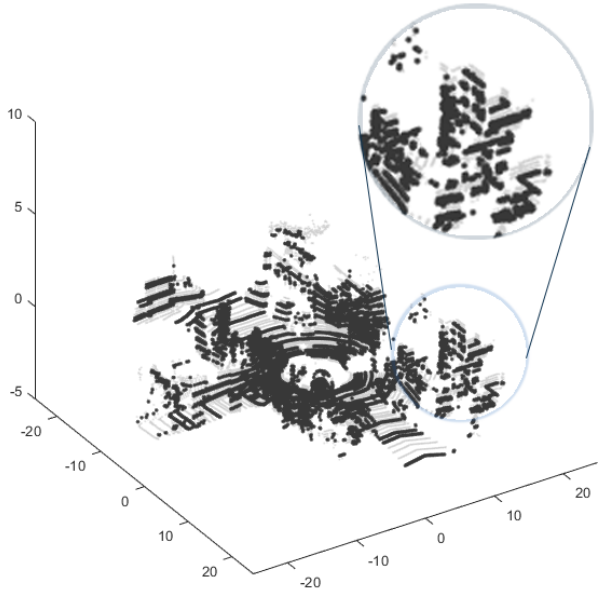

Figura 3.3: Nube de puntos resultante primer método

de la nube de puntos identifican estos haces como círculos que localizan el techo del coche, o el suelo en torno a 1 metro dell vehículo. Por lo que se tomaron los haces desde el 19, descartando la información de 2 haces intermedios, hasta el 64 (los haces utilizados son 19, 22, 25, 28, 31, ..., 61, 64). Finalmente se tiene un rango de ángulo de elevación de 2 a -18 grados en la nube de puntos de 16. Esta información se adaptará más a un LiDAR de 16 haces real (Velodyne VLP-16) ya que el rango de ángulo de elevación de este LiDAR es de 15 a -15 grados. Se puede visualizar en la figura 3.4, siendo la nube en gris la nube de 64 haces, y la nube en color negro la de 16 haces.

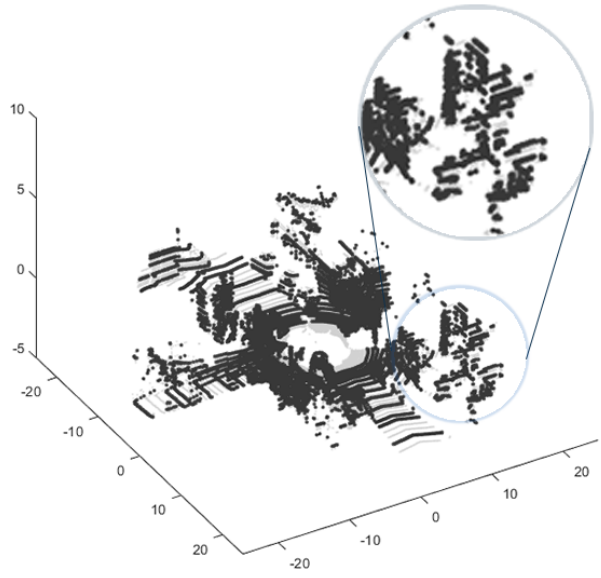

Figura 3.4: Nube de puntos resultante segundo método

De estos dos métodos, el que mejor ha funcionado cualitativamente es el primero, es por esto que en los resultados solo se muestran imágenes con esta dataset.

3.3 Procesamiento de los datos en Matlab

Se tienen 2 tipos de datos con los que entrenaremos la red, los obtenidos desde CARLA y los obtenidos a partir de KITTI, los cuales se preparan de la misma manera. La única diferencia es que los datos de CARLA son archivos en formato PLY y los datos de KITTI son matrices en Matlab, de las cuales se comentó su adaptación en el apartado anterior.

Luego de tomar los datos del simulador tenemos dos carpetas de nubes de puntos en formato PLY, una con los datos del LiDAR de 16 haces y otra con el de 64, o en el caso de los datos de KITTI, dos matrices por cada nube de puntos ya leída, una de 16 haces y la otra de 64. Para entrenar las redes GAN se necesitan imágenes, por lo que se utilizará Matlab para la creación del dataset, esto es, convertir las nubes de puntos en imágenes en escala de grises en formato PNG y renombrarlas para organizarlas.

Las imágenes que se generarán son mapas de cuadricula polar, o PGM por sus siglas en inglés. Esta es una representación de un espacio tridimensional en una matriz. En esta representación se toma un punto (origen), y a partir de él se toma un rango del ángulo de elevación el cual se representa en las filas, se toman los 360 grados del plano horizontal y se representan en las columnas. Mediante la variable de colorse representará la profundidad a la que se encuentra cada punto. La diferencia que hay entre este tipo de representación y las imágenes de profundidad convencionales es que en las imágenes de profundidad no se abarcan los 360 grados en torno a la cámara, sino solo el área de enfoque de ésta.

Para esta aplicación, primero se convertirán los datos de coordenadas cartesianas a polares, en donde el origen será la posición del sensor LiDAR en el coche, los ángulos de elevación y azimut vienen dados por las coordenadas, y el radio representa la profundidad, por lo que a partir de organizar esa información se convierten fácilmente en imágenes. Las ecuaciones 3.1, 3.2 y 3.3 fueron las utilizadas para la transformación, teniendo en cuenta la figura 3.5.

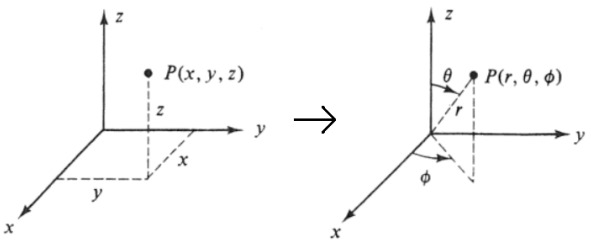

Figura 3.5: Diagrama de coordenadas cartesianas y polares.

$$x = r \cdot \operatorname{sen}\phi \cdot \cos\theta \tag{3.1}$$

$$y = r \cdot \operatorname{sen}\phi \cdot \sin\theta \tag{3.2}$$

$$z = r \cdot \cos\phi \tag{3.3}$$

Es necesario comentar que se pierden puntos en el paso de nubes de puntos a imágenes, ya que las imágenes están divididas en pixeles, y se trabaja con resoluciones bajas, por ejemplo, 256x256. Al momento de clasificar los puntos de los 360 grados en 256 columnas, disminuye la resolución de los datos obtenidos directamente del LiDAR, siendo el error de un sensor de este tipo en torno a 0,5 grados. La información que se pierde no es algo que para los fines para los que se utilizarán los datos sea de suma importancia,

ya que los algoritmos que trabajan en la percepción de objetos suelen basarse en la coplanaridad de los puntos, ya que la densidad de los puntos es uniforme en la nube de puntos de un LiDAR. en la figura 3.6 se puede apreciar claramente que la distribución de puntos sigue siendo uniforme, y los puntos no cambian de posición ni se altera la geometría que siguen, solo cambia la cantidad de puntos que forman cada haz.

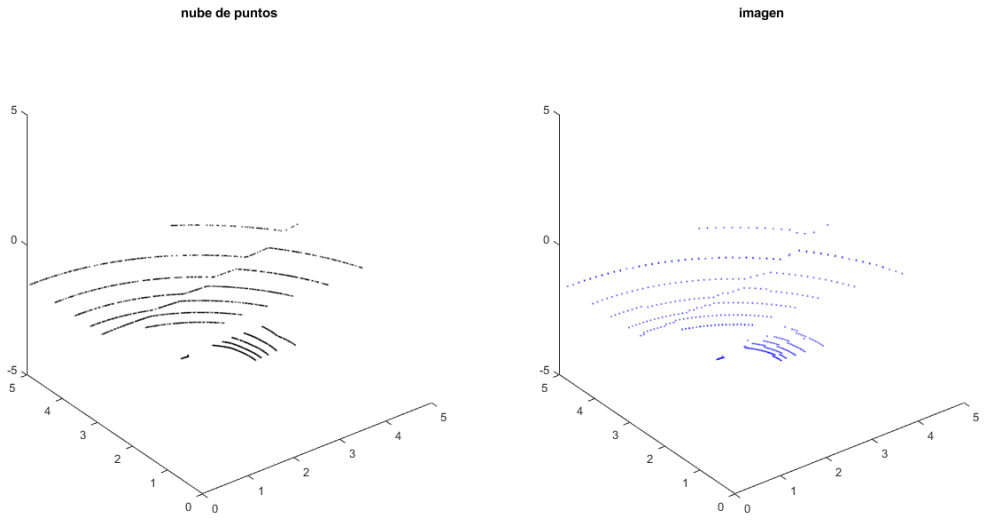

Figura 3.6: Pérdida de puntos en la conversión de nubes de puntos a imágenes

Cada haz de una nube generada en CARLA tiene aproximadamente 1160 puntos, con la configuración utilizada, y cuando se transforman a imagen pueden tener un maximo de 256 puntos por haz. En la ecuación 3.4 se calcula el porcentaje de puntos que se pierden por haz.

$$Perdida = 100 - (\frac{256}{1160} \cdot 100) = 79{,}93\,\% \tag{3.4}$$

En el dataset de KITTI cada haz tiene una media de 1380 puntos, el porcentaje de pérdida de puntos por haz es el que se calcula en la ecuación 3.5.

$$Perdida = 100 - (\frac{256}{1380} \cdot 100) = 81{,}14\,\% \tag{3.5}$$

El programa que se ha creado permite modificar el tamaño de la imagen creada, para tener facilidad al momento de hacer pruebas con diversos tipos de datos, ya que la red con la que trabajamos está diseñada para crear imágenes que sean similares al ojo humano, esto quiere decir que la red discriminadora evalúa la similitud en aspectos cualitativos, pero el objetivo es representar distancias y aunque las imágenes sean similares al ojo humano, si no conservan los aspectos cuantitativos no son útiles para la aplicación. Por lo que se han creado múltiples datasets con variaciones en tamaños de imagen, en rango de profundidad, en ángulo de azimut tomado para la creación de la imagen, para encontrar la mejor manera de modelar las nubes de puntos en este formato para realizar la aplicación deseada. Algunos de los ejemplos utilizados están presentes en la figura 3.7.

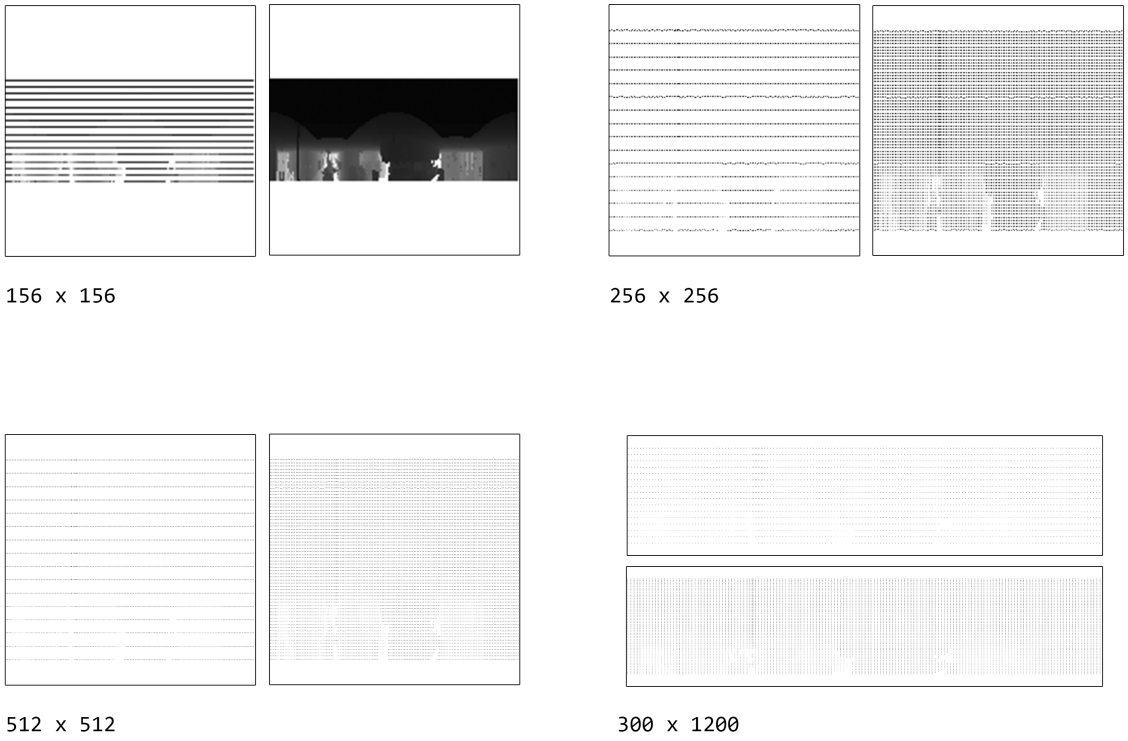

Figura 3.7: Variaciones en las imágenes que componen el dataset

Como existe un programa que convierte de nube de puntos a imagen, también hay otro que hace el mismo proceso, pero en sentido contrario, transformando las imágenes a nubes de puntos. Este también fue desarrollado en Matlab para el proceso de evaluación y el estudio de los resultados, para observar la similitud ente salida deseada y salida obtenida. Utiliza las mismas ecuaciones que el anteriormente comentado, estas son las ecuaciones 3.1, 3.2 y 3.3.

A su vez, se ha creado un programa que ordena las imágenes en carpetas, nombradas de la manera que requiere la red, para facilitar su manejo al momento de realizar el entrenamiento.

3.4 Elección del algoritmo de Deep Learning

Al momento de realizar el aumento de resolución de los datos, se puede plantear de maneras muy diferentes. Por ejemplo, trabajar directamente con nubes de puntos, que son matrices de dimensión Nx3, siendo N el número de puntos de la nube, o transformando la información de nube de puntos a una vista de pájaro, trabajando con la perspectiva desde planta, o realizar la transformación de nubes de puntos a imágenes de profundidad o similar, entre otros. El problema de trabajar con nubes de puntos directamente es que es un área en la cual el aumento de resolución en ellas es algo que no está muy desarrollado y un área donde hay poca información. En su contraparte, el aumento de resolución en imágenes está presente desde hace mucho tiempo y es muy común, utilizándose en todas las áreas, desde trabajar con imágenes de la cámara de un móvil hasta trabajar con imágenes astronómicas tomadas con múltiples telescopios de agujeros negros. Por esta razón se decidió trabajar con imágenes, son métodos bien desarrollados y con gran capacidad, a pesar de ser sencillos de utilizar.

Para el desarrollo de esta aplicación se eligieron dos redes neuronales tipo GAN [7] en una primera instancia, de las cuales, luego de ser evaluadas, se elegiría una para llevar a cabo la tarea.

Estas redes son dos proyectos separados, pero se han obtenido de un repositorio de GitHub [10], en el cual se encuentran los dos sistemas juntos, y se puede interactuar con ambos a partir de una base de opciones entre las cuales se puede elegir entre la red GAN que se desea utilizar. Además de ser dos redes GAN muy potentes y con muchas utilidades para diversas tareas, este repositorio facilita la configuración de las redes en todos sus aspectos, sin la necesidad de ser un experto en redes neuronales ni en ningún lenguaje de programación. Permite modificar las redes en todos sus aspectos, modificar cada uno de sus parámetros, el tipo de redes que serán la generadora y la discriminadora, el número de sus capas, el Learning Rate, etc. También permite modificar todos los parámetros de entrenamiento, prueba y validación de la red utilizada, sea cual sea, permitiendo modificar el número de iteraciones, el preprocesamiento de los datos, el entrenamiento por lotes, la normalización de lotes, la ganancia a la entrada, etc. Por último, añade un sistema de visualización del entrenamiento, mostrando las gráficas de los errores de cada red interna a la GAN, los resultados de cada iteración junto con la entrada y la salida esperada, y permite modificar la frecuencia con la que sean tomados estos datos. Todos estos datos son mostrados con el paquete Visdom, y la ventana que se visualiza durante el entrenamiento a arece en la figura 3.8. En ella se observa en tiempo real mientras la red neuronal se entrena, los errores de las redes internas a la GAN, y una muestra de la entrada, el groundtruth y la salida de la ultima iteración.

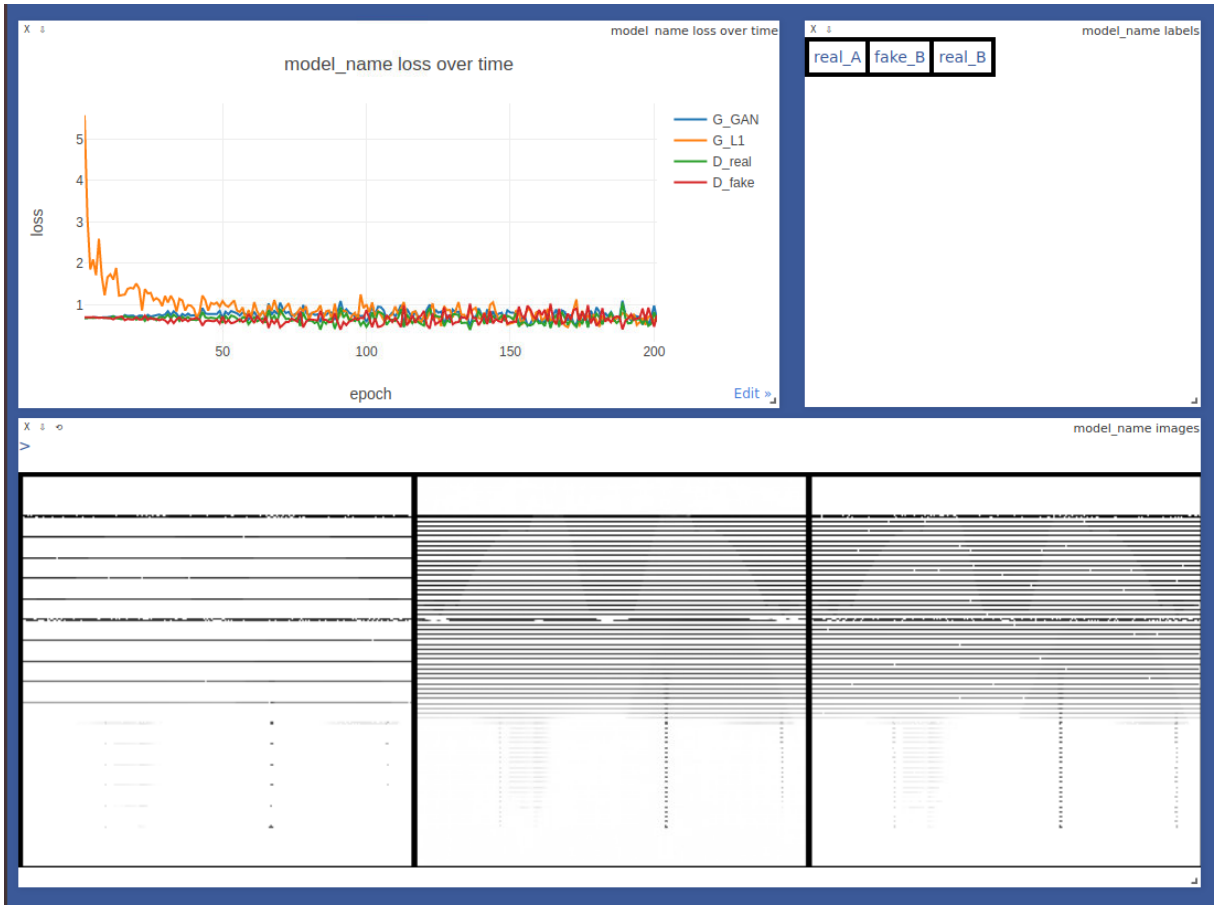

Figura 3.8: Interfaz de entrenamiento con Visdom

Para la evaluación de qué red sería la más apropiada para la aplicación que se está diseñando, se han hecho diversos entrenamientos en ambas redes, con la misma configuración y con el mismo dataset cada vez, y se han comparado los resultados. Los datasets constaban de 100 parejas de imágenes y se entrenaron

en 200 iteraciones, sin entrenar por lotes. Una vez obtenidas las imágenes de salida, se compararon con las imágenes deseadas, no solo como imágenes, sino también transformadas a nubes de puntos. En este momento se define si la hipótesis planeada de poder convertir nubes de puntos de 16 haces a 64 haces a través de imágenes PGM era cierta o no.

Al final de dicha evaluación, se decidió utilizar la red pix2pix, ya que en comparación con la red cycleGAN, la pix2pix presenta mucho menos desenfoque en sus resultados de salida, algo que es clave para el sistema que se está desarrollando. Tambien se comenta que la red pix2pix se entrena de manera más rápida, ya que la cycleGAN presenta 4 redes, 2 generadoras y 2 discriminadoras, una pareja para cada sentido de transformación. Así se obtienen nubes de puntos más precisas y más cercanas a las deseadas.

Una vez que se tuvo la red pix2pix como la más acertada, se ha estudiado esta red en sus diferentes configuraciones, cambiando parámetros como las redes utilizadas, tanto la generadora como la discriminadora, y probando su funcionamiento con los diferentes datasets.

De esta manera se logró identificar que, con esta red, se logra el funcionamiento esperado con la siguiente configuración:

Sensores LiDAR:

- Rango: 25 m
- Frecuencia:1 Hz
- Puntos/segundo (64 haces): 120000
- Puntos/segundo (16 haces): 30000

Imágenes:

- Tamaño: 256x256 pixeles
- Rango de azimut: 360 grados
- Rango de elevación: -35 a 15 grados
- Rango de profundidad: 0 a 25 metros

Pix2pix:

- Tamaño del lote: 1
- Canales de entrada: 1 (escala de grises)
- Canales de salida: 1 (escala de grises)
- Learning Rate: 0.0002
- Número de iteraciones: 100
- Número de iteraciones con decaimiento del Learning Rate: 100
- Red discriminadora: pixel
- Número de capas de red discriminadora: 3
- Número de filtros en la primera capa convolucional de la red discriminadora: 64

- Red generadora: unet 256
- Número de filtros en la última capa convolucional de la red discriminadora: 64
- Preprocesamiento de datos: No

3.5 Entrenamiento y preparación de la red pix2pix

Para el entrenamiento final de la red se ha creado una simulación de 2000 segundos, de la cual se han obtenido 2000 parejas de muestras para realizar el entrenamiento. Se ha entrenado con los parámetros que se habían hallado óptimos.

El entrenamiento se realizó con el dataset de imágenes creadas en CARLA, con las opciones y comandos que facilita el repositorio de GitHub, explicados en el manual de usuario.

Luego de tener las redes generadora y discriminadora entrenadas, se prepara la red para ser utilizada como un elemento del sistema. Lo que se hace es fijar todas las opciones variables en las opciones en las que se han obtenido los mejores resultados. Luego aparte se crean en Python los mismos programas que se crearon en Matlab de conversión de nube de puntos a imagen y viceversa para utilizarlos a la entrada de la red y a la salida de la misma.

La red al momento de formar parte del sistema completo funcionará recibiendo un dato de entrada que es una nube de puntos en formato de matriz, se supone que ya se ha convertido del formato que lo envía el LiDAR a un array en Python. En el momento que se recibe, se convierte a una imagen en PNG codificada en 8 bits (profundidad de color desde 0 a 255) y una vez que tenemos la imagen, se pasa por la red entrenada donde se convierte a una imagen de las mismas características, pero representando una nube de puntos de un LiDAR de 64 haces. A la salida de la red, se toma esa imagen y se convierte a una nube de puntos en formato PLY, la cual luego se transforma a un array el cual es la salida final. Todo este sistema ha sido diseñado en Python y se muestra un diagrama de él en la figura 3.9.

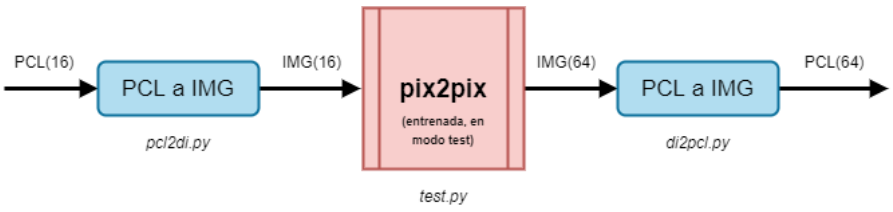

Figura 3.9: Diagrama de funcionamiento del sistema final

3.6 Implementación con ROS

Como se ha explicado, este sistema que se ha diseñado solo será un paso intermedio entre el sensor y el software de percepción de un vehículo autónomo, que además funcionará todo en tiempo real. La herramienta que se ha utilizado es ROS para todas las comunicaciones internas, desde la salida del LiDAR, recibiendo los datos en el ordenador del coche y utilizando esos datos en la percepción de los objetos.

Para la toma de datos se creará un suscriber de ROS, el cual estará suscrito al topic al cual el sensor LiDAR envia las nubes de puntos, y se creará un callback, esto es una función que se ejecutará cada vez que llegue un dato del sensor LiDAR. En esta función se define la conversión de nube de puntos de un sensor de 16 haces a nube de puntos de un sensor de 64 haces, es decir que el sistema diseñado realizará su funcionamiento en este punto.

Para la salida de la nube de puntos se crea un publisher, el cual publicará la nube de puntos de salida en otro topic, del cual se leerá ya la nube de puntos de 64 haces, al cual puede estar suscrito el software que se encargará de la percepción.

Todo este sistema está graficado en la figura 3.10 a continuación, donde tambien se muestra con que tipo de datos se trabaja en cada punto del sistema.

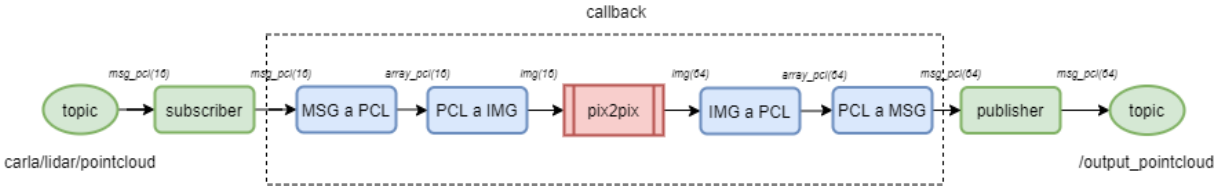

Figura 3.10: Diagrama de funcionamiento en ROS

Capítulo 4

Resultados

4.1 Introducción

Los resultados que se muestran en el siguiente capítulo, son los resultados obtenidos a la salida de la red pix2pix. Se han realizado múltiples entrenamientos, variando la configuración del dataset, esto se refiere tanto a sensores como a imágenes. Se presentan en concreto 4 de los muchos modelos obtenidos, los cuales se identificaron como los que presentaban resultados más útiles al momento de su uso para aplicaciones de percepción del entorno. Dos de ellos son pruebas realizadas sobre datos obtenidos en simulación, en CARLA, y los otros dos son obtenidos a partir del dataset que proporciona KITTI. La diferencia entre los dos entrenamientos realizados con cada conjunto de datos es la profundidad de los LiDARs, un entrenamiento fue realizado con 25 metros de rango y el otro con 50 metros.

Al inicio se han realizado pruebas con una profundidad de 250 metros, obteniendo resultados para nada útiles, ya que si tenemos en cuenta que la profundidad de color de las imágenes con las que trabaja la red es de 8 bits (valores de 0 a 255), un error de una unidad en el color de la imagen se traduce como un error de aproximadamente un metro en el espacio real. Al disminuirse en diez veces la distancia, el error también disminuye diez veces. Con 25 metros de rango se han obtenido los mejores resultados, pero es poco útil trabajar con un sensor LiDAR para distancias tan cortas. Una de sus ventajas es ser preciso a grandes distancias, y utilizar 25 metros sería desaprovechar uno de los beneficios más grandes que tiene el utilizar sensores LiDAR. Por esta razón se realizaron los mismos estudios para una distancia de profundidad de 50 metros, que ya es algo más razonable, y una distancia útil para trabajar con detección de objetos, tracking, etc., todo en el ámbito de la precepción.

Es necesario comentar que previamente a los ensayos realizados con las configuraciones mostradas, primeramente, se tuvieron que hacer diversas evaluaciones de las datasets utilizadas y de las configuraciones de las redes entrenadas. A base de prueba y error se ha llegado a estas configuraciones mostradas, siendo ellas las más aptas para ser utilizadas en la percepción. Las características de las imágenes resultantes que se utilizaron para determinar si un modelo en su primera prueba era válido o no, fue meramente cualitativo, observando el parecido entre las imágenes de entrada y de salida. Aspectos a verificar eran el suavizado entre los pixeles de la imagen, si las imágenes presentaban degradado en el pase de un haz al otro, las líneas de cada haz bien diferenciadas, entre otros aspectos que se determinan de manera visual. Se debe comentar que las imágenes que se evaluaron fueron muy variadas, y son determinantes aspectos como la cantidad de pixeles de un mismo color, la continuidad de color en la imagen, etc.

Por ejemplo, los datos de 512x512 pixeles, donde los pixeles en blanco en la imagen eran la mayoría, y no se visualizaban líneas continuas, la red procesaba ese pixel en un tono gris o gris oscuro como un

error en la imagen. No olvidemos que son redes diseñadas para obtener datos visualmente correctos, es decir que se enfocan en conseguir resultados que para el ojo humano sean correctos, y al aportar tan poca información, un pixel en una imagen de resolución media, no era posible obtener resultados útiles. Es correcto afirmar que, a mayor resolución en la imagen, también será menor el error de los ángulos de azimut y elevación, pero a su vez esto afecta en la manera en que la red interpreta la imagen y, por lo tanto, a la salida obtenida. A su vez, con imágenes de 156x156 pixeles, o menos, se ha logrado obtener una imagen continua, sin espacios en blanco entre filas ni columnas. Esto permitía que la red interprete mejor la información que se proporcionaba, pero afectaba a la resolución en ángulos de azimut y elevación, disminuyéndola. Por eso, se ha optado por trabajar con un tamaño intermedio, el cual es un tamaño estándar para las redes convolucionales que funcionan como redes generadoras dentro de una red GAN, este tamaño es 256x256 pixeles.

Además de observar las imágenes, cuando el aspecto visual era apto para ser capaz de tener información útil, se realizó la conversión de dichas imágenes a nubes de puntos, donde también se realizó un proceso de elección de los datasets y las configuraciones más idóneas a partir de sus aspectos cualitativos, pero en este caso, también se hizo un análisis de sus aspectos cuantitativos. Se ha utilizado una métrica de error que determina la distancia mínima media entre los puntos de la nube de 64 haces esperada y la obtenida.

La manera en la que se presentan los resultados es haciendo diferentes comparativas visuales, tanto entre imágenes como entre nubes de puntos, estas últimas, presentándolas en diferentes perspectivas, para así tener una idea más clara de la similitud entre la salida esperada y la obtenida, o también entre los datos de entrada y los datos de salida, y la creación de esos nuevos puntos generados por el algoritmo de Deep Learning. Se ha utilizado Matlab para presentar los resultados, debido a la facilidad con que permite obtener las gráficas, calcular el error entre ellos, presentar las imágenes, etc. Para cada uno de los cuatro modelos se presentan las mismas graficas comparativas, para poder no solamente comprobar los resultados de un modelo en particular, sino también entre los diferentes modelos.

4.2 Estudio del Error

Para el cálculo del error se ha utilizado el método KNN (K-Nearest Neighbors), o método de los k vecinos más cercanos, este algoritmo encuentra los k puntos más cercanos al punto que se está estudiando. Lo utilizaremos con k=1, lo que significa que asigna el elemento de la matriz A que se está estudiando, al elemento más cercano perteneciente a la matriz B, siendo A y B matrices de 3 dimensiones. Lo que se hace es medir las distancias entre cada punto de la nube de puntos de 64 haces obtenida de la red y su punto asociado por el algoritmo KNN de la nube de 64 haces esperada (groundtruth), y realizar la media entre todas las distancias. Es una métrica de error que permite conocer cuál es la distancia media entre el punto obtenido y el real más cercano.

No hay una métrica determinada de error para nubes de puntos y es difícil hacer un estudio del error entre las nubes de puntos, ya que no hay un punto de la nube obtenida asociado a cada punto de la nube esperada, la cantidad de puntos de cada una es diferente. Es diferente a estudiar el error en una imagen, donde se puede estudiar pixel a pixel la diferencia de color, es decir que es una malla donde para cada punto en la imagen obtenida a la salida, hay un punto asociado en la imagen esperada, el cual es una misma posición en el espacio para ambas imágenes. Estudiando la diferencia de la profundidad de color en cada pixel se puede identificar cual es el radio esperado que se tiene para unos ángulos de azimut y de elevación determinados, y cuál es el radio obtenido después de la conversión.

No es posible hacer el mismo estudio mencionado con imágenes en un espacio de tres dimensiones continuo, como lo son las nubes de puntos con las que trabajamos. Hay una opción la cual es voxelizar

las nubes de puntos, esto es crear una malla de 3 dimensiones y clasificar los puntos de las nubes en los puntos de la malla más cercanos a ellos. Esto sería una manera de llevar la solución al mismo campo que con las imágenes, discretizando el espacio y estudiando cada punto por sí mismo.

Pero la solución que se ha llevado cabo permite realizar un estudio del error sin perder resolución al discretizar nuevamente los datos con los que se trabaja, ya que para cada punto en la nube obtenida hay solo un punto el cual es el más cercano a este, y esa distancia es la que se toma como error entre el valor esperado y el estimado. El algoritmo busca entre todos los puntos que hay en la nube de puntos del espacio real (nube de puntos esperada) cual es el punto más cercano, es decir, a que distancia está el punto de la nube estimada por la red de la posición que debería tener en el espacio real.

4.3 Pruebas Experimentales

4.3.1 Setup de pruebas

Cada una de las pruebas que se mostrarán a continuación en los siguientes apartados, han sido realiazadas con las mismas cantidades de muestras, las cuales se dividen en datos de entrenamiento, de test y de validación.

Los datos de entrenamiento lo componen 100 parejas de muestras, formadas por una nube de puntos de 16 haces y una nube de puntos de 64 haces, que en el proceso se convertirán a imagenes antes de entrenar la red. Es un número razonable para las pruebas, pero pequeño para el entrenamiento final. De estos 100 datos, un 10 % de ellos, elegidos aleatoriamente, se han utilizado para validación. Luego se tienen 10 muestras, las cuales no estan presentes en los datos de entrenamiento, con los cuales se probó la red. Esta información comentada se resume en la siguiente tabla.

	Entrenamiento	Validación	Test
Pruebas	100	10	10
Entrenamiento final	2000	200	100

Tabla 4.1: Número de muestras para cada fase

Los datos tomados de cada entrono se muestran a continuación, en la figura 4.1 para CARLA y en la figura 4.2, acompañados de una imagen del entorno para tener una idea sobre el medio en que se tomaron tanto los datos de KITTI como los de CARLA.

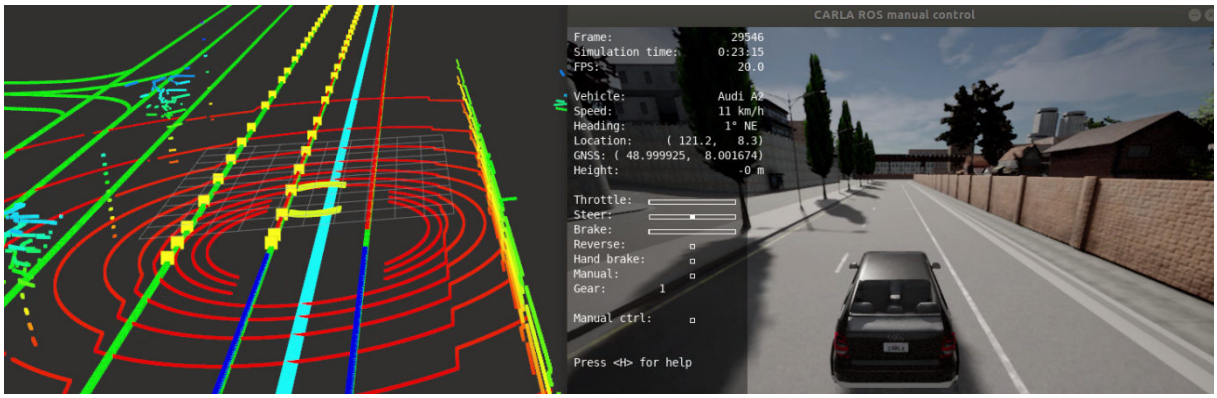

Figura 4.1: Datos tomados desde CARLA

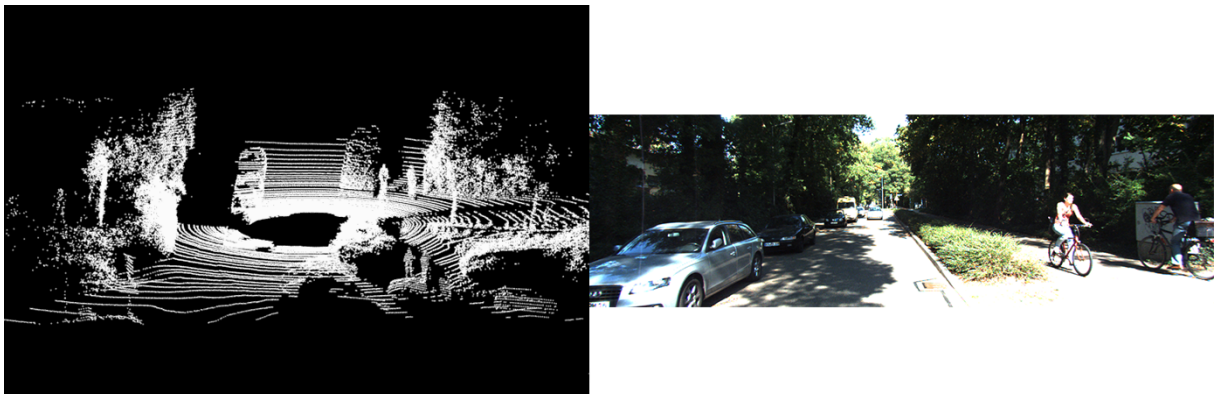

Figura 4.2: Datos tomados desde KITTI

4.3.2 Pruebas con dataset CARLA, 25 metros de rango

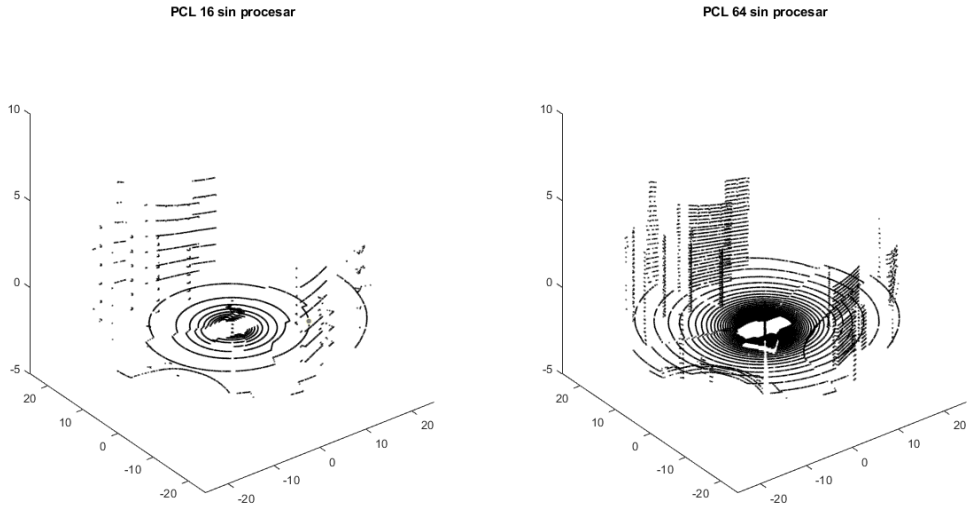

Figura 4.3: Nubes de puntos de entrada sin procesar

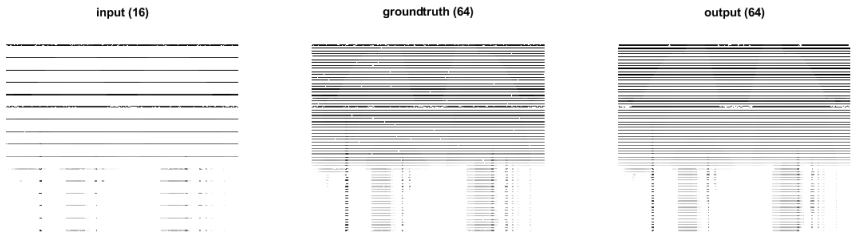

Figura 4.4: Imágenes de entrenamiento e imágen resultante

En la figura 4.3 se muestra los datos iniciales, los que entran a la red. A la izquierda, la nube de puntos de 16 haces, y a la derecha, la nube de puntos de 64 haces. Para esta primera prueba se tiene un rango de 25 metros. Luego en la figura 4.4 se muestran las imágenes correspondiente a las nubes de puntos anteriores, pero además tambien se muestra la imagen que generó la red neuronal a partir de la imagen de nube de 16 haces. Se observa en las imágenes como en los haces inferiores, los cuales representan los haces superiores de la nube de puntos, hay muchos espacios en blanco, esto significa que no existe información en estos puntos. Esto se debe a que el rango no es suficiente para obtener la información de esos puntos que se encuentran más allá de los 25 metros.

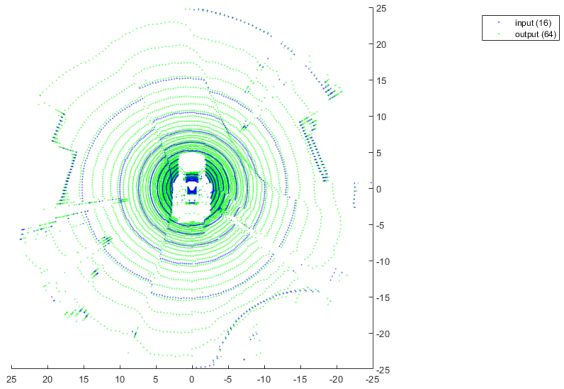

Figura 4.5: Comparación de imágenes de entrada y salida de la red pix2pix, vista de pájaro

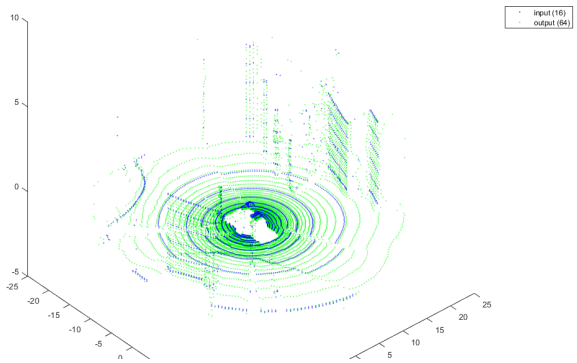

Figura 4.6: Comparación de imágenes de entrada y salida de la red pix2pix, perspectiva isométrica

En las figuras 4.5 y 4.6 se visualizan desde dos perspectivas diferentes la nube generada a partir de la imagen de entrada a la red y la nube de puntos generada a partir de la imagen de salida de la red, que esta ultima será la salida de nuestro sistema total. La nube de puntos azul es lo que entra a la red neuronal en formato de imagen, y la nube en color verde es la que sale. Se observa que los haces de 64 son continuos y uniformes, los sectores donde la red no es capaz de interpretar correctamente la información es donde los haces no son continuos, y se da un salto de una superficie a otra, donde se observa que la red intenta suavisar esa transición.

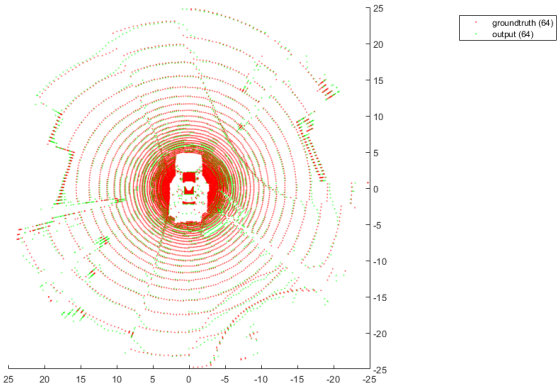

Figura 4.7: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, vista de pájaro

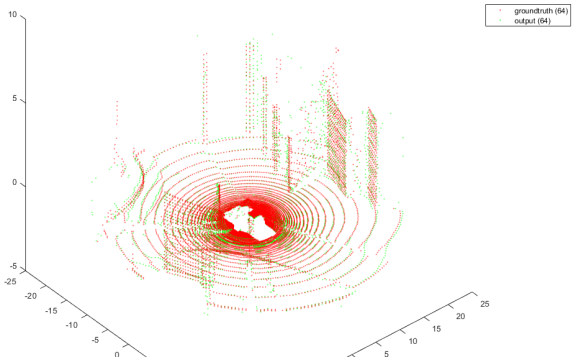

Figura 4.8: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, perspectiva isométrica

En las figuras anteriores, 4.7 y 4.8, se compara el grountruth de la red con la imagen de salida de la red. La nube de color verde, idealmente, deberia ser igual a la nube de color rojo. Se observa en este caso el error que comete la red neuronal en la interpolación en los puntos donde los haces no son continuos. Pero los haces generados son continuos, no presentan grandes irregularidades con respecto a los haces que se esperan.

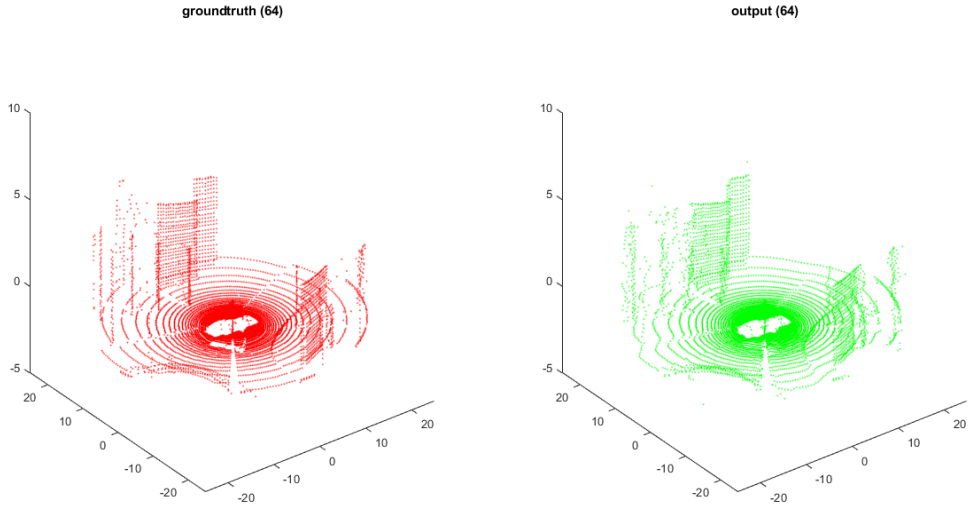

Figura 4.9: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, perspectiva isométrica, separados

En las figuras anteriores, 4.7 y 4.8, se compara el groundtruth de la red con la imagen de salida de la red. La nube de color verde, idealmente, deberia ser igual a la nube de color rojo. Se observa en este caso el error que comete la red neuronal en la interpolación de los puntos donde los haces no son continuos. Pero los haces generados son continuos, no presentan grandes irregularidades, con respecto a los haces que se esperan. En la figura 4.9 se representan las mismas nubes pero separadas, se puede observar la similitud entre ellas.

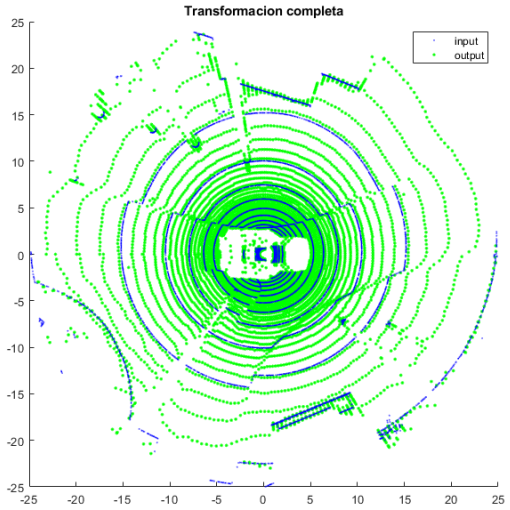

Figura 4.10: Comparación de nube de puntos de entrada al sistema y nube de puntos de salida del sistema, vista de pájaro

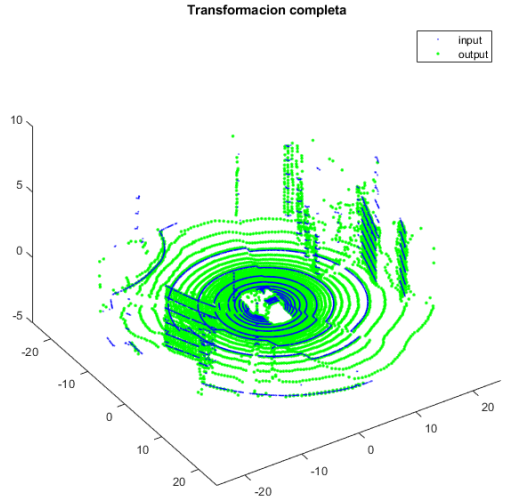

Figura 4.11: Comparación de nube de puntos de entrada al sistema y nube de puntos de salida del sistema, perspectiva isométrica

En estas dos figuras anteriores, 4.10 y 4.11, se puede observar la nube de puntos de entrada, antes de convertirla a imagen, es decir, con una densidad de puntos mayor, y la nube de puntos de de salida, mostrando los datos de los que se parte y los datos obtenidos.

4.3.3 Pruebas con dataset CARLA, 50 metros de rango

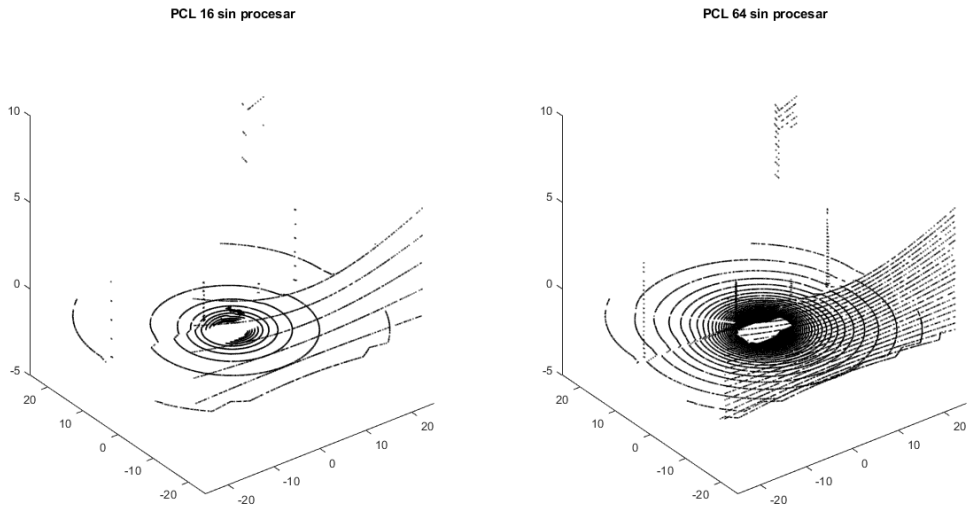

Figura 4.12: Nubes de puntos de entrada sin procesar

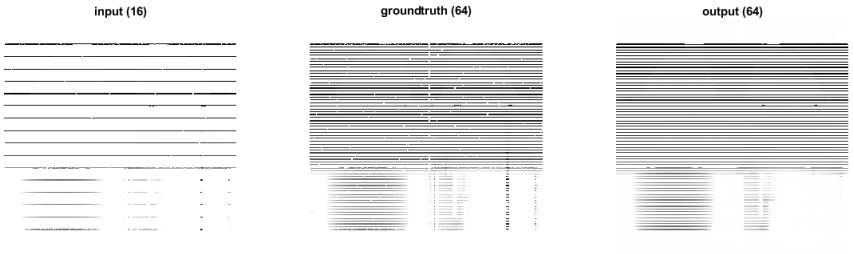

Figura 4.13: Imágenes de entrenamiento e imágen resultante

En la figura 4.12 se muestra los datos iniciales sin procesamiento, esta vez se tiene un radio de 50 metros, los datos se salen de la gráfica mostrada, que solo representa de -25 a 25 metros en los ejes x e y. A la izquierda, la nube de puntos de 16 haces, y a la derecha, la nube de puntos de 64 haces. Luego en la figura 4.13 se muestran las imágenes correspondiente a las nubes de puntos anteriores, pero además tambien se muestra la imagen que generó la red neuronal a partir de la imagen de nube de 16 haces. Se observa en las imágenes como en los haces inferiores, los cuales representan los haces superiores de la nube de puntos, ahora el espacio en blanco, es decir, sin información, es menor. Ahora con un mayor rango, el LiDAR es capaz de obtener información mas lejana, pero a su vez, la resolución en profundidad dsiminuye a la mitad.

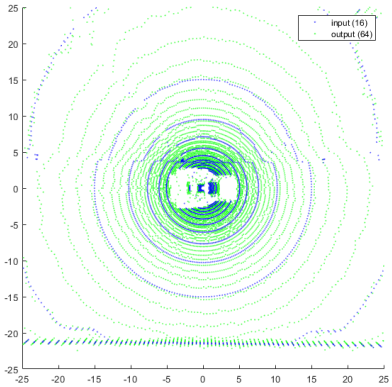

Figura 4.14: Comparación de imágenes de entrada y salida de la red pix2pix, vista de pájaro

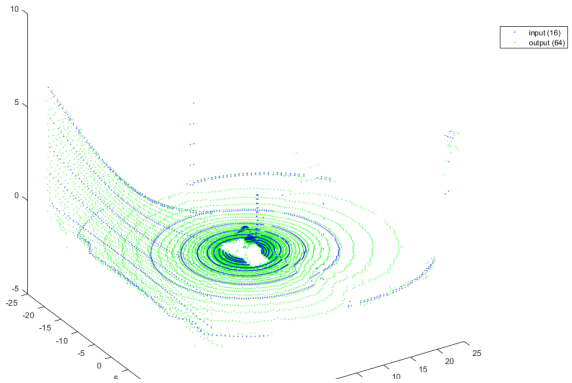

Figura 4.15: Comparación de imágenes de entrada y salida de la red pix2pix, perspectiva isométrica

En las figuras 4.14 y 4.15 se visualizan desde dos perspectivas diferentes la nube generada a partir de la imagen de entrada a la red y la nube de puntos generada a partir de la imagen de salida de la red, que esta ultima será la salida de nuestro sistema total. La nube de puntos azul es lo que entra a la red neuronal en formato de imagen, y la nube en color verde es la que sale. Se observa que los haces de 64 son continuos y pero se ven menos uniformes, las superficies son menos claras, y los cambios de profundidad en un haz se ven mucho mas afectados.

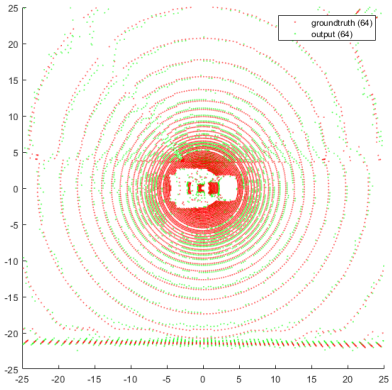

Figura 4.16: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, vista de pájaro

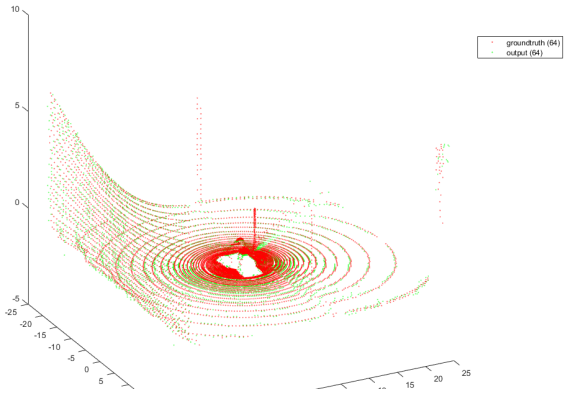

Figura 4.17: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, perspectiva isométrica

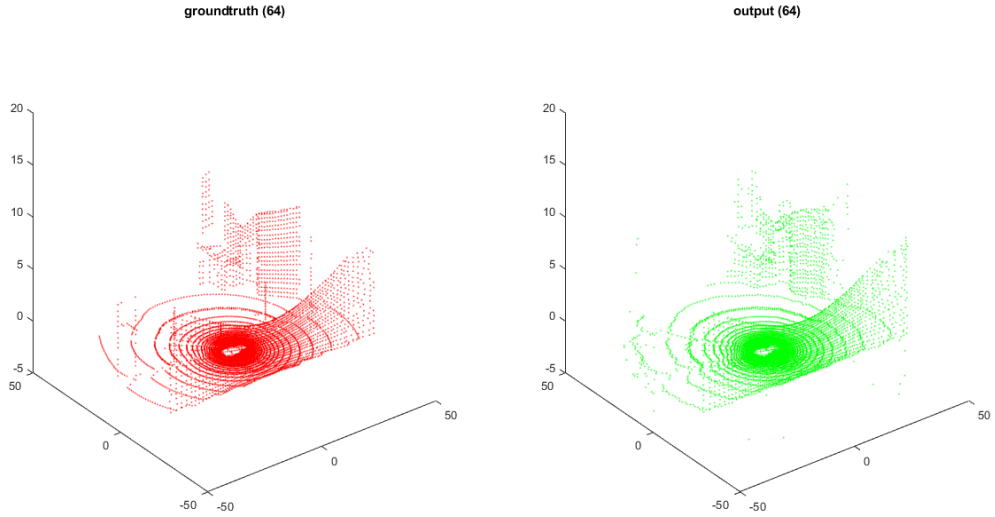

Figura 4.18: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, perspectiva isométrica, separados

En las figuras 4.16 y 4.17, se compara el grountruth de la red con la imagen de salida de la red. La nube de color verde, idealmente, deberia ser igual a la nube de color rojo. En esta prueba, se puede observar como los haces son menos exactos, como son menos uniformes y varían más que en el caso de un LiDAR con un rango de 25 metros. Pero los haces generados son continuos, presentan más irregularidades que el las pruebas con 25 metros de rango, pero así mismo, son similares y se asemejan a la nube esperada. En la figura 4.18 se ve claramente la nube de salida mas irregular que la nube generada con el dataset de 25 metros de rango, pero aún asi, sigue conservando la información de las superficies, siendo cualitativamente similar a la nube esperada.

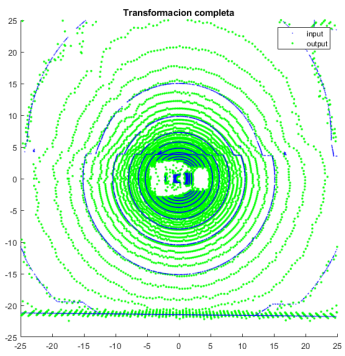

Figura 4.19: Comparación de nube de puntos de entrada al sistema y nube de puntos de salida del sistema, vista de pájaro

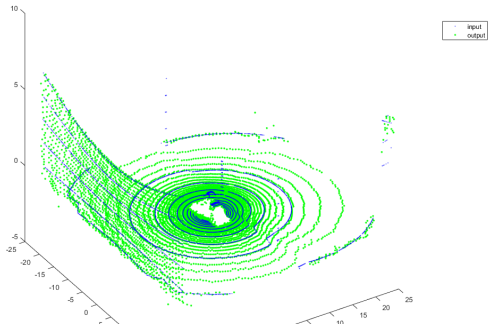

Figura 4.20: Comparación de nube de puntos de entrada al sistema y nube de puntos de salida del sistema, perspectiva isométrica

En las dos figuras anteriores, 4.19 y 4.20, se puede observar la nube de puntos de la que se parte, la cual tiene una densidad de puntos mayor, y la nube de puntos de de salida, mostrando los datos de los que se parte y los datos obtenidos. Cualitativamente, esta transformación es peor que la prueba anterior (datos de 25 metros de rango), pero se tiene información del doble de distancia que en el caso anterior.

4.3.4 Pruebas con dataset KITTI, 25 metros de rango

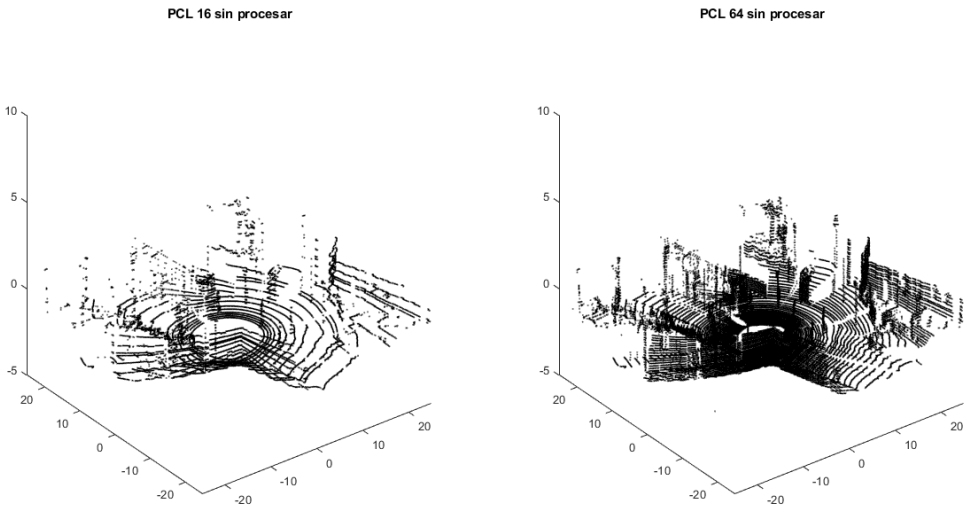

Figura 4.21: Nubes de puntos de entrada sin procesar

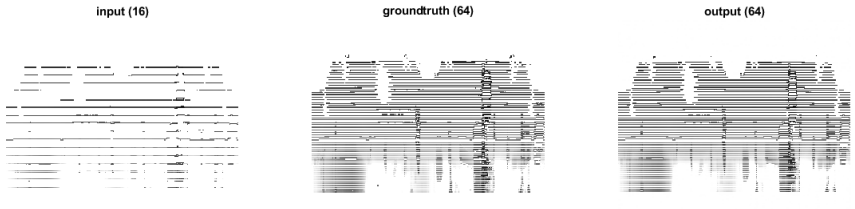

Figura 4.22: Imágenes de entrenamiento e imágen resultante

Estos son datos de un sensor real, en la figura 4.21 se observa como estos datos son mas irregulares, menos claros y precisos, muy diferentes a los datos obtenidos en CARLA. A su vez, las imágenes generadas, que se pueden ver en la figura 4.22, son también más irregulares que las imágenes que se presentan en los datasets obtenidos desde CARLA. Esto no es un parámetro que afecte al funcionamiento de la red, la red de igual manera es capaz de trabajar con estos datos.

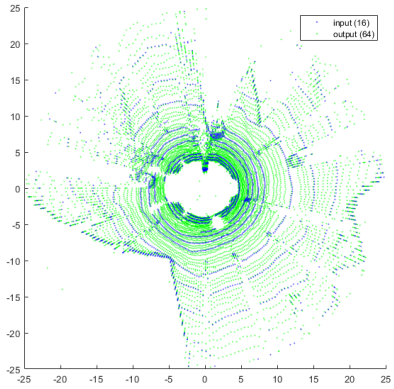

Figura 4.23: Comparación de imágenes de entrada y salida de la red pix2pix, vista de pájaro

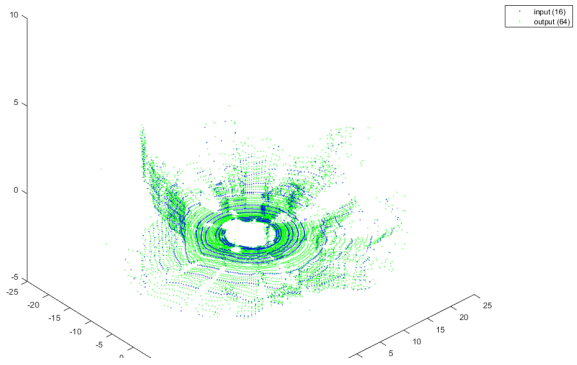

Figura 4.24: Comparación de imágenes de entrada y salida de la red pix2pix, perspectiva isométrica

Se observan en las figuras 4.23 y 4.24 dos perspectivas diferentes la nube generada a partir de la imagen de entrada a la red y la nube de puntos generada a partir de la imagen de salida de la red, que esta ultima será la salida de nuestro sistema completo. La nube de puntos azul es la entrada a la red neuronal en formato de imagen, y la nube en color verde es la nube de salida de la red neuronal. A pesar de que los haces de entrada son mas irregulares, vemos que la red es capaz de transformar correctamente los datos, realizando una interpolación a simple vista correcta.

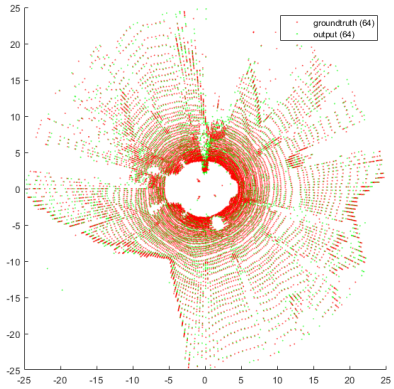

Figura 4.25: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, vista de pájaro

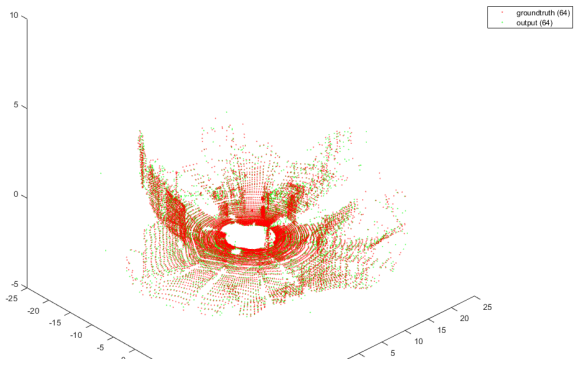

Figura 4.26: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, perspectiva isométrica

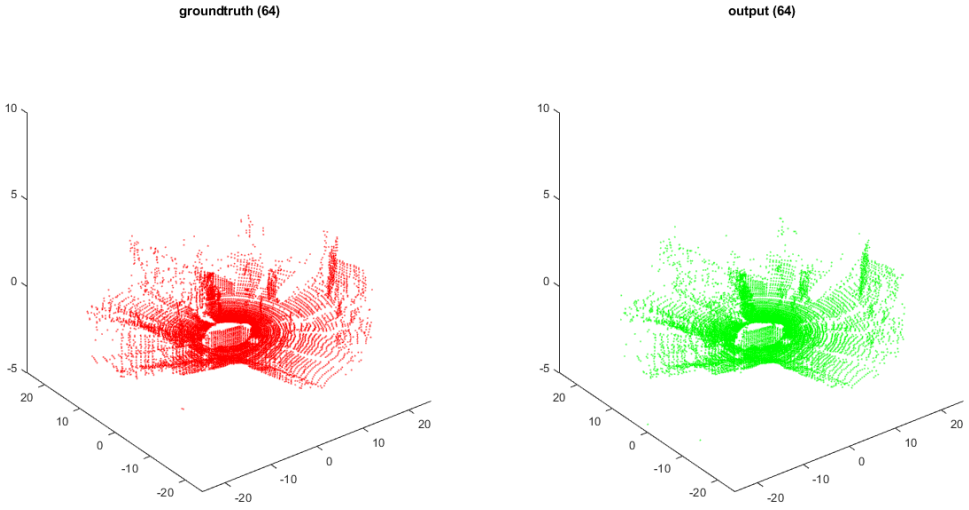

Figura 4.27: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, perspectiva isométrica, separados

En las figuras 4.25 y 4.26 se comparan la nube de la imagen de la salida esperada y la de la salida obtenida. Se observa que aunque los datos de entrada son irregulares y parecen menos ordenados, los datos de salida de la red logran asemejarse en gran manera a la salida esperada, con gran precisión. En la figura 4.27 donde se comparan de manera separada, se observan prácticamente iguales.

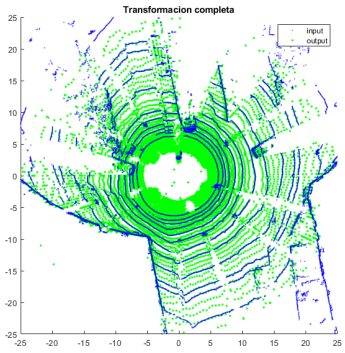

Figura 4.28: Comparación de nube de puntos de entrada al sistema y nube de puntos de salida del sistema, vista de pájaro

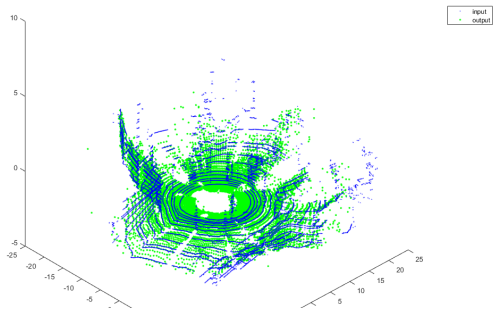

Figura 4.29: Comparación de nube de puntos de entrada al sistema y nube de puntos de salida del sistema, perspectiva isométrica

Estas últimas dos imágenes, figura 4.28 y figura 4.29, muestran la transformación completa, donde la nube azul es la entrada al sistema, y la nube verde es la salida del sistema. Lo más importante a destacar de estas pruebas es que utilizando datos reales, sujetos a irregularidades, ruidos, variaciones al momento de toamr los datos, etc, no afectan al funcionamiento del sistema de aumento de resolución en número de haces que se ha desarrollado.

4.3.5 Pruebas con dataset KITTI, 50 metros de rango

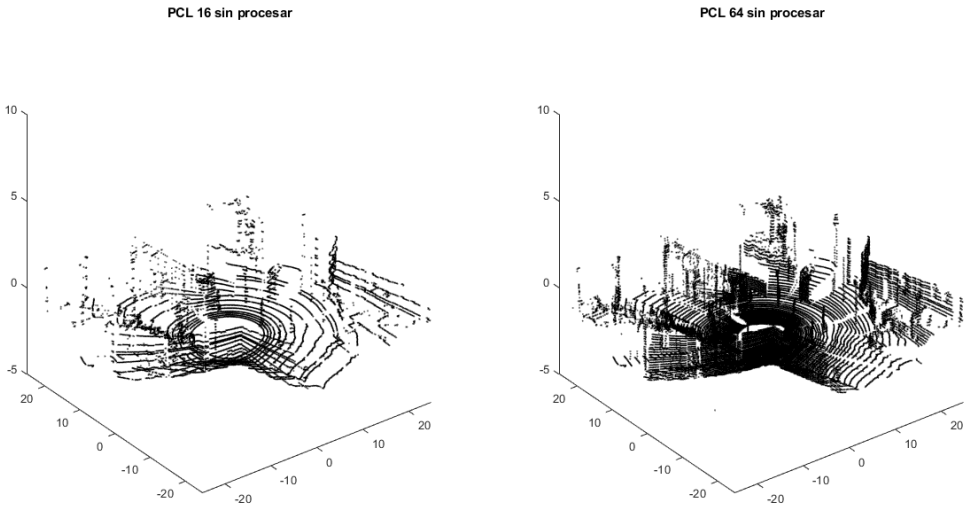

Figura 4.30: Nubes de puntos de entrada sin procesar

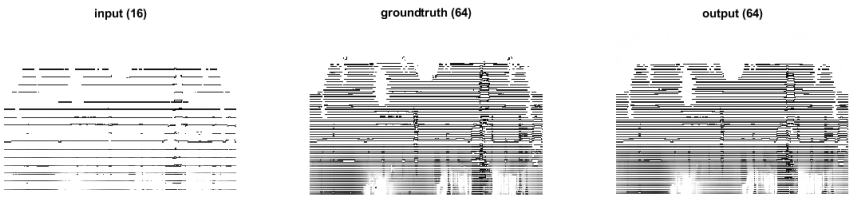

Figura 4.31: Imágenes de entrenamiento e imágen resultante

Recordar que estos datos son exactamentes los mismos que para la prueba anterior (25 metros), pero en este caso se ha reducido el rango a 50 metros. En la figura 4.30 se observa la irregularidad y poca precisión ya que son datos reales, muy diferentes a los datos obtenidos en CARLA. Las imágenes generadas, que se pueden ver en la figura 4.31, que dejan poco espacio sin información, al tratarse de un entorno real y a un sensor real. Las imágenes son también más irregulares que las imágenes que se presentan en los

datasets obtenidos desde CARLA. A pesar de eso, la red es capaz de poder trabajar con estos datos y no afectan a su desempeño.

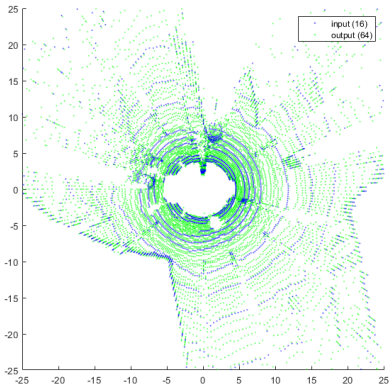

Figura 4.32: Comparación de imágenes de entrada y salida de la red pix2pix, vista de pájaro

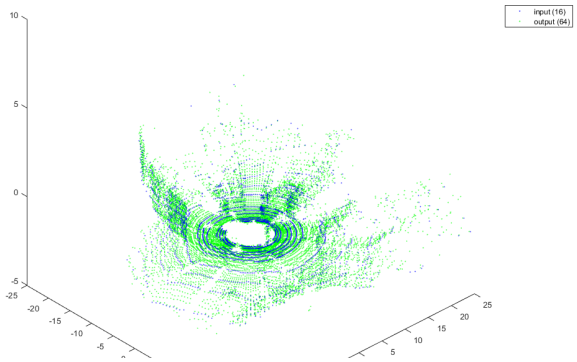

Figura 4.33: Comparación de imágenes de entrada y salida de la red pix2pix, perspectiva isométrica

Se observan en las figuras 4.32 y 4.33 dos perspectivas diferentes la nube generada a partir de la imagen de entrada a la red y la nube de puntos generada a partir de la imagen de salida de la red, que esta ultima será la salida de nuestro sistema completo. La nube de puntos azul es la entrada a la red neuronal en formato de imagen, y la nube en color verde es la nube de salida de la red neuronal. La forma básica se mantiene, pero hay que admitir que la nube de puntos se ve mucho mas desordenada que en los casos anteriores, la suma de los errores propios de un sensor real y los que agrega la transformación se hacen presentes.

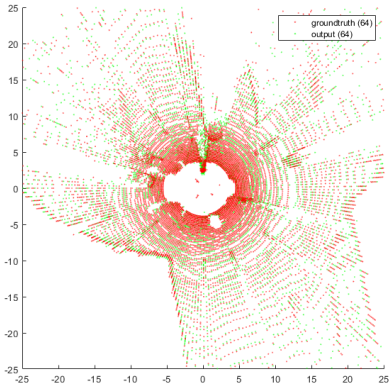

Figura 4.34: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, vista de pájaro

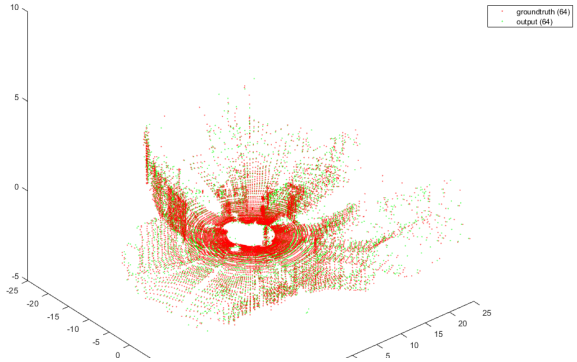

Figura 4.35: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, perspectiva isométrica

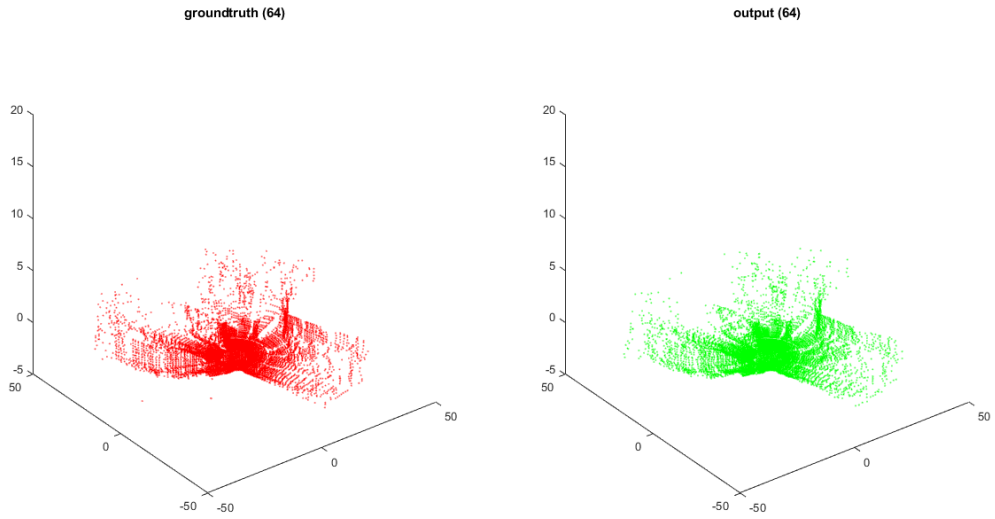

Figura 4.36: Comparación de imágenes de salida obtenida y salida esperada de la red pix2pix, perspectiva isométrica, separados

En las figuras anteriores, figuras 4.34 y 4.35, se comparan la nube de la imagen de la salida esperada y

la de la salida obtenida. Se observa claramente la diferencia, los haces coinciden muy poco. Son cercanos y siguen el mismo patrón, pero a pequeña escala se nota la diferencia. En la figura 4.36 donde se comparan de manera separada, se observa la similitud, pero se nota que es mucho mas irregular la salida obtenida en esta prueba que en las anteriores.

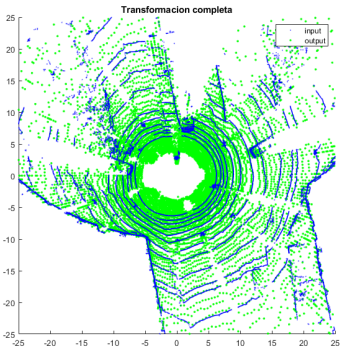

Figura 4.37: Comparación de nube de puntos de entrada al sistema y nube de puntos de salida del sistema, vista de pájaro

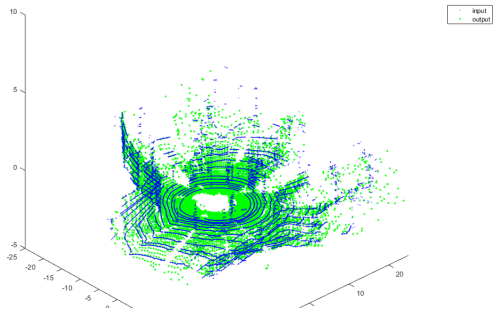

Figura 4.38: Comparación de nube de puntos de entrada al sistema y nube de puntos de salida del sistema, perspectiva isométrica

Estas últimas imágenes, figura 4.37 y figura 4.38, muestran la transformación completa del sistema, donde la nube azul es la entrada al sistema, y la nube verde es la salida del sistema. Como en el caso anterior, prueba con el dataset de KITTI y con un rango de 25 metros, con estos datos el sistema es funcional. Los datos obtenidos en este caso son menos precisos, como se habia de esperar, la nube de puntos obtenida esta mas dispersa, pero se sigue asemejando a la nube de puntos de entrada.

4.3.6 Errores

A continuación, se muestra un diagrama completo del sistema utilizado durante el entrenamiento y las pruebas, donde se detalla los datos en cada momento del proceso y se muestra su identificación, la cual luego se utilizará en las tablas.

En la entrada al sistema se tiene un bloque que representa el dataset utilizado, del cual tenemos simplemente nubes de puntos, los cuales son datos que no se pueden utilizar directamente en la red, por lo que se deben de procesar. Los elementos PCL(16) y PCL(64) son la pareja de nubes de puntos que entrenarán a la red, siendo PCL(16) la nube de puntos de 16 haces la cual se transformara a una nube de 64 haces, siendo PCL(64) el resultado ideal esperado, o groundtruth. El procesamiento lo que hace es

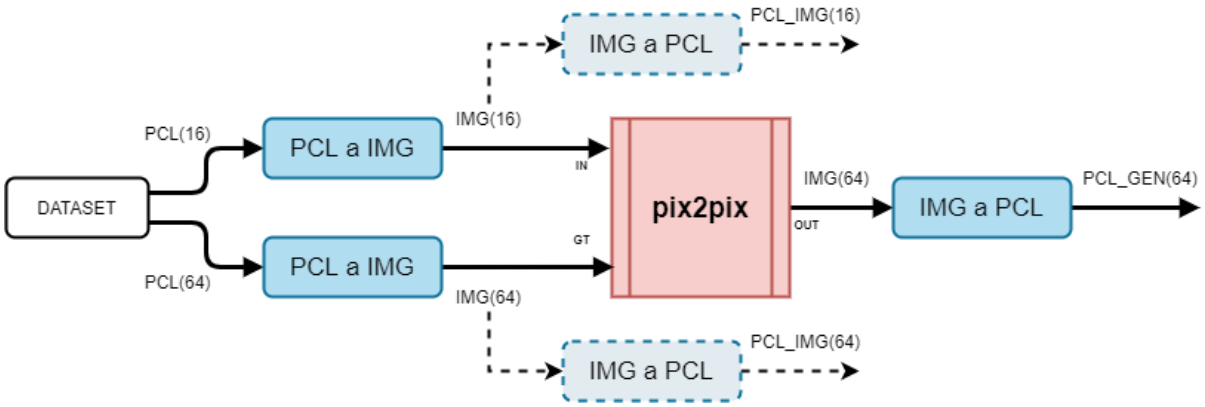

Figura 4.39: Diagrama de entrenamiento y prueba de la red pix2pix

convertir la nube de puntos a imagen, para que pueda ser aplicado a la red pix2pix, donde obtenemos los datos IMG(16) y IMG(64), que son las nubes de puntos PCL(16) y PCL(64) reorganizando la información de las nubes de puntos en formato PGM, es decir en imágenes. Luego aparecen en líneas punteadas los datos PCL_IMG(16) y PCL_IMG(64), que aparecen de esta manera porque no afectan al sistema, solo se sacan estos datos para evaluar la pérdida de puntos en la conversión de nube de puntos a imágenes PGM. A la salida de la red pix2pix tenemos la imagen genera por la red neuronal, identificada como IMG(64) out, la cual luego para poder ser utilizada se debe procesar nuevamente, realizando ahora la conversión inversa. Es así como tenemos a la salida el dato PCL_GEN(64), la cual es una nube de puntos apta para ser utilizada en otras aplicaciones, siendo esta la salida de nuestro sistema.

La métrica del error utilizada fue comentada en la sección 4.2, en donde se comenta que ese valor que se ha ordenado en las tablas a continuación hace referencia a distintas medidas en imágenes y en nubes de puntos. En nubes de puntos este valor es la distancia media que existe entre cada punto de la nube analizada y el más cercano a él, perteneciente a la nube de puntos tomada como referencia para estudiar este error. En imágenes es el mismo valor, la distancia media entre un punto de la imagen y otro punto de la imagen tomada como referencia, pero estos puntos son el mismo pixel para cada imagen, y la distancia sería siempre una distancia radial.

Dataset	Rango	PCL(16) vs PCL_IMG(16)	PCL(64) vs PCL_IMG(64)
CARLA	25	0.1153	0.0787
CARLA	50	0.2011	0.1550
KITTI	25	0.3225	0.0964
KITTI	50	0.3926	0.2118

Tabla 4.2: Error cometido al realizar la transformación de nube de punto a imagen (metros)

En esta primera tabla se muestra el error cometido al momento de convertir las nubes de puntos a imágenes, tanto la de 16 haces, la cual es el dato de entrada al sistema, como la nube de puntos de 64 haces, la cual será utilizada por la red para ser entrenada y a su vez, será utilizada en las pruebas, siendo esta imagen la imagen ideal que se desea tener a la salida. Se hace el estudio entre la nube de puntos inicial y la nube de puntos obtenida a partir de la imagen generada, es decir, una conversión y una desconversión.

Se observa como la distancia media siempre es superior en las nubes de puntos de rango mayor. Esto es algo trivial, sabiendo que la profundidad de color con la que se han codificado las imágenes de

entrenamiento es de 8 bits (esto es que los posibles valores que puede tomar un pixel es de 0 a 255) . Por lo que, sabiendo que la resolución es la mínima variación de la magnitud medida que da lugar a una variación perceptible de la indicación correspondiente, se tiene lo siguiente:

$$Resolucion_{25m} = \frac{25\,\mathrm{m}}{255} = 0.098\,\mathrm{m} \tag{4.1}$$

$$Resolucion_{50m} = \frac{50\,\mathrm{m}}{255} = 0.196\,\mathrm{m} \tag{4.2}$$

Cabe recalcar que en la dataset de CARLA se mantiene un error menor en esta conversión, esto se debe a que los datos tomados en el simulador tienen poco o nulo efecto las perturbaciones, ruidos, etc, que podrían afectar en las irregularidades de los datos reales.

En cuanto a los datos de KITTI no hay que olvidar que los datos de 16 haces son datos generados por nosotros, por lo que se observa en la medida del error que también se suma el error anteriormente cometido en pasar de una nube de 64 haces a una de 16.

En la siguiente tabla se presenta el error entre la nube de puntos de 64 haces generada a partir de la imagen y la nube de puntos de salida.

Dataset	Rango	PCL_IMG(64) vs PCL_GEN(64)
CARLA	25	0.0904
CARLA	50	0.2241
KITTI	25	0.1110
KITTI	50	0.1608

Tabla 4.3: Error cometido entre las nubes de la salida esperada y la salida obtenida (metros)

Como era de esperar, el error utilizando la nube antes de ser procesada como imagen siempre es más grande que el error donde se tiene en cuenta la nube de puntos ya procesada. Se observa que en ambos dataset la distancia mínima media ronda los 10 centímetros para el rango de 25 metros y 20 centímetros para el de 50 metros. Esto se puede definir como el error aportado por la conversión en la red pix2pix.

Dataset	Rango	IMG(64)gt vs IMG(64)out	IMG(16)in vs IMG(64)out
CARLA	25	0.2731	-
CARLA	50	0.2788	-
KITTI	25	0.1964	0.1107
KITTI	50	0.2254	0.0908

Tabla 4.4: Error cometido en la conversión de imágenes (metros)

Esta tabla presenta los errores entre la entrada y la salida de la red neuronal, es decir, en imágenes. Primeramente, comentar que se estudia únicamente la información contenida en los haces de la imagen de entrada a la red, no se tienen en cuenta los puntos en blanco de la imagen, en los cuales no hay información porque son puntos fuera del rango del LiDAR. También decir que cuando se evalúa la imagen de 16 haces, son solo los 16 haces en los cuales la imagen tiene información contrastados con estos mismos haces en la imagen de 64 haces generada. No olvidar que cada fila de pixeles en la imagen es un ángulo en la nube de puntos.

Con el estudio de la imagen de 16 haces y la de 64 lo que se busca es verificar que la información

que aportan los 16 haces sigue presente en la de 64 haces. En la imagen de CARLA no se pudieron tomar datos de esta manera, ya que por la división de los ángulos de los haces que hace el simulador, no coinciden en ángulo los 16 haces dentro de la nube de 64. De manera contraría, en la dataset que se creó a partir de KITTI, los 16 haces fueron tomados de la nube de 64, por lo que coinciden los ángulos.

Para los datos de CARLA el error fue mayor en ambos datasets de diferentes rangos. Esto se debe en parte a que los datos obtenidos desde CARLA son menos realistas, y por otra parte a que en los datos de KITTI en la nube de puntos de 16 haces tenemos la información exacta de 16 de los 64 haces a generar por la red. Por la manera en que genera los haces el simulador CARLA, los haces de la nube de 16 y los de la nube de 64 no son coincidentes, sino solo el primero y el último. En el dataset de KITTI todos los puntos en la nube de 16 haces esta presente en la nube de 64, por la manera en la que se creó. La nube de 16 fue creada a partir de la nube de 64. Teóricamente, este error es posible disminuirlo a partir de modificaciones en el entrenamiento de la red neuronal.

Finalmente se presenta el error total del sistema, se comparan la nube de 64 haces directamente tomada del sensor o de la simulación, con la nubede 64 haces que se ha generado, pasando por la conversión a imagen y la interpolación para aumentar su resolución en la red neuronal.

Dataset	Rango	PCL(64) vs PCL_GEN(64)
CARLA	25	0.1116
CARLA	50	0.2470
KITTI	25	0.1454
KITTI	50	0.2463

Tabla 4.5: Error cometido respecto a la nube de puntos de salida final (metros)

Los valores del error, es decir la distancia media entre los puntos de la nube generada y los puntos mas cercanos en la nube de puntos de 64 haces inicial, son muy similares en ambos datasets. Aproximadamente unos 25 centímetros en las pruebas con un rango de 50 metros para ambas datasets, y para 25 metros de rango, unos 11 centímetros para el dataset de CARLA y 14 centímetros para KITTI. No afecta en gran manera con que datos se trabaje, el error es similar en ambos datasets.

Capítulo 5

Conclusiones

En este apartado se comentan las conclusiones obtenidas a partir del trabajo de investigación realizado sobre el aumento de resolución de los datos obtenidos por un LiDAR de 16 haces, convirtiéndolos mediante algoritmos de Deep Learning en datos de un LiDAR de 64 haces. Se debe tener en cuenta que es un campo poco estudiado, ya que se suele optar por adaptar los mecanismos de percepción que trabajan con LiDARs de 64 haces a LiDARs de 16 haces, y no hacerlo de la manera que se presenta en este trabajo, adaptando los datos.

5.1 Conclusiones

Se ha partido de la hipótesis de que era posible generar nubes de puntos correspondientes a un sensor LiDAR de 64 haces haciendo uso de algoritmos de Deep Learning, solamente utilizando los datos provenientes de un LiDAR de 16 haces. A partir de las pruebas y ensayos realizados, se ha podido comprobar que si es posible realizar esta conversión.

Hablando de manera general, los resultados obtenidos son buenos, y demuestran que es posible hacer uso del Deep Learning para aumentar la resolución de nubes de puntos, pasando de una cantidad de haces determinada a otra mayor, conservando las características que posee un sensor de este tipo.

Se ha logrado reducir la distancia mínima media entre un punto estimado al real de en torno a los 10 centímetros, lo cual es una gran precisión, pero también hay que decir que esto es para una distancia de 25 metros, lo cual reduce mucho las capacidades de un sensor LiDAR.

En resumen, este sistema necesita ser mejorado para su implementación en aplicaciones reales y no desaprovechar las capacidades del sensor LiDAR. Además se deberia realizar un entrenamiento con datos del sensor LiDAR a utilizar, con su debida configuración.

Para que el sistema pueda tener un funcionamiento casi ideal, es decir, para lograr la mejor conversión de nubes de 16 haces a nubes de 64, con la mayor precisión posible, son muchos aspectos los que se deben mejorar, los cuales todos se deben tener en cuenta.

En primer lugar, los datos son la base del Deep Learning, sin datos no es posible entrenar una red, y con datos malos no es posible entrenar correctamente una red. Los datos que se utilizaron para este trabajo no fueron los mejores datos de los que se ha podido disponer.

Con respecto a los datos obtenidos de CARLA, debemos comentar que es un simulador muy útil porque abarca y cubre muy bien todas las áreas de la percepción, posee todos los sensores, brinda imágenes muy realistas, proporciona datos de IMU muy reales, pero tiene deficiencias en los sensores LiDAR. Las nubes

de puntos proporcionadas son muy poco similares a las reales, las representaciones de los vehículos en la nube de puntos son similares a un prisma, completamente rectangular, teniendo una perspectiva muy poco acertada a lo que serían datos reales de un sensor LiDAR. En sus últimas versiones se está buscando mejorar las nubes de puntos. De ser así, sería una gran ventaja y permitiría mejorar este sistema, ya que, en simulación es la única manera en la que se pueden tener dos sensores con diferentes características en un mismo punto, lo que es esencial para el entrenamiento de esta red.

Con respecto a los datos de KITTI, este es un dataset de un modelo determinado de LiDAR de 64 haces, lo cual determina mucho la utilidad que se podría darle. A diferencia de CARLA, donde se puede configurar a gusto el sensor LiDAR y obtener todas las muestras que se quieran, en todos los escenarios que se desee, con el dataset de KITTI se tiene el inconveniente de que el LiDAR es un modelo único, con una configuración única, son muestras determinadas, por lo que es un dataset que no permite ser flexibles. Hay que adaptarse a los datos, estando muy limitados al momento de convertirlos en un dataset para lo cual no fue creado, como por ejemplo en la aplicación diseñada, donde se transforman los datos de 64 haces a 16 para luego reconvertirlos con Deep Learning en datos de 64 haces.

Para desarrollar el sistema ideal se necesitarían, primeramente, nubes de puntos realistas. Esto se podría lograr teniendo dos sensores LiDAR, uno de 16 haces y otro de 64 haces, que estén prácticamente en el mismo punto, lo cual es imposible, pero podría implementarse una estructura que tenga un sensor encima del otro, y asi poder tomar datos que sean tomados prácticamente desde la misma posición. O a través de simulación, por ejemplo, en CARLA, con una nube de puntos mejorada, configurando los parámetros de los sensores de tal manera que sean exactamente iguales a los LiDARs reales (parámetros como rango de ángulo de elevación, frecuencia de giro, cantidad de puntos por segundo, rango) para que sea lo más acertado a los datos de un LiDAR real. Una vez obtenidos los datos, sería importante aumentar la resolución de las imágenes, tanto en tamaño como en profundidad de color, esto puede resolverse codificando la profundidad en 16 bits, o en 3 canales (RGB) en vez de solo en uno.

Con respecto al entrenamiento de la red, y como conseguir el mejor o más óptimo, se determina de manera estocástica, es decir, en cuanto a las redes neuronales, no hay una manera que sea la óptima, o la mejor, si no que se basa mucho en los resultados que se hayan obtenido empíricamente.

El mayor defecto de los resultados obtenidos es que son poco precisos. Son exactos, ya que los haces generados se acercan a los deseados, pero los puntos están dispersos respecto a cómo deberían estar de manera ideal.

Son muchas las maneras en que perdemos precisión de las medidas iniciales tomadas por el sensor. Este trabajo se ha desarrollado y se han obtenidos resultados utilizables para el fin que se requerían con nubes de puntos que abarcan un radio de 25 metros alrededor del coche. Esta distancia es una distancia que con nubes de puntos transformadas en imágenes PNG codificadas en un canal de 8 bits, se obtiene una resolución en la distancia desde el vehículo hasta el punto de alrededor de 10 centímetros. La precisión de un LiDAR es de milímetros, y con la conversión completa que se realiza se pierde en parte, siendo la precisión una de las ventajas más grandes de este tipo de sensores.

Otra forma en la que se pierde información es al discretizar la información y encasillarla en una rejilla para convertirla en imágenes. Es decir, cada pixel de la imagen es un ángulo de elevación y un ángulo de azimut, y la información que haya entre pixel y pixel se pierde. A su vez, los sensores LiDAR de 64 proporcionan una nube de puntos de alrededor de 120.000 puntos, y con la resolución que se está trabajando (256x256), se podría tener un máximo de 65.000 puntos, que en la práctica jamás se obtendrán, porque cada haz de laser será una fila de pixeles, por lo que habrá filas intermedias vacías, lo que deja alrededor de 12.000 puntos en una nube de 64 haces. Se pierden el 90 % de los puntos que tendría una nube de un LiDAR de 64 haces. Esto no es igual que decir que se pierde el 90 % de la información, ya que

lo que se hace en realidad es reorganizar toda esa información, pero en un espacio discreto. Por ejemplo 10 puntos de una nube de puntos de entrada pueden estar asociados en un mismo pixel, al final lo que se pierde es densidad de puntos, pero la información que aportaban esos 10 puntos en la nube, es la misma información que aporta un pixel en la imagen, la cual utilizará el algoritmo de percepción para realizar su trabajo.

5.2 Trabajos Futuros

A continuación se presentan algunas de las líneas propuestas por las que se puede continuar este trabajo, ya que como se ha comentado, es una técnica con gran potencial y útil para obtener grandes resultados a partir de sensores LiDAR económicos.

- Realización una mejora en la resolución de las imágenes de entrenamiento, obteniendo aumento de precisión en este mismo sistema, a partir de la codificación de las imágenes en un formato que aumente la resolución de la distancia radial en las nubes de puntos. Por ejemplo, codificar las imagenes PNG en 16 bits o realizar una codificación en RGB (24 bits).
- Mejora del método de conversión para aumentar el rango conservando la precisión utilizando dos redes neuronalales, utilizando cada una para un intervalo de distancia diferente, por ejemplo, una red para convertir de 0 a 25 metros y otra para convertir de 25 a 50 metros.
- Estudio de la mejora en la detección de objetos utilizando este sistema de aumento de resolución, preparado y entrenado para un modelo de LiDAR de 16 haces en particular, por ejemplo el Velodyne VLP-16 puck, y aplicarle nuestro sistema de aumento de datos.
- Integración de este sistema a un sistema de percepción ya desarrollado y estudiar la mejora en la detección de objetos, o tracking, y comparar con el uso de un LiDAR de 16 haces y un LiDAR real de 64 haces.

Capítulo 6

Manual de Usuario

6.1 Requisitos previos

Para trabajar con estas redes neuronales de manera cómoda y eficiente se recomienda utilizar un ordenador con capacidades medias-altas de procesamiento. El proyecto desarrollado permite correr las redes en el CPU, pero es muy recomendable utilizar GPU, con posibilidad de trabajar con CUDA. Además de esto, que se tengan todos los drivers de la tarjeta gráfica instalados y configurados, y que sean compatibles la version del driver con la version de CUDA.

Esta red se instalará en un entorno de Linux, se recomienda trabajar en Ubuntu, en la versión 18.04 LTS. Se necesita instalar PyTorch, en su versión 0.4 o mayor. También se ofrece en el repositorio otros proyectos los cuales utilizan otras bibliotecas como por ejemplo Tensorflow, Keras, Lasagne, entre otros. Es necesario tener instalado en el sistema Python 3 o mayor.

Los paquetes de Python y versiones a instalar son los siguientes:

- torch 1.4.0
- torchvision 0.5.0
- dominate 2.4.0
- visdom 0.1.8.8

6.2 Instalación de red Pix2pix

Para instalar la red que se utilizará, simplemente hace falta clonar el repositorio de GitHub del proyecto [10]:

`git clone https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix cd pytorch-CycleGAN-and-pix2pix`

Con esto, se descargará todo lo necesario para realizar el entrenamiento de la red.

6.3 Crear dataset en CARLA

Para preparar los datos de entrenamiento a partir de CARLA, se debe ejecutar el programa creado para ello ("dataset_creation.py"), configurando los sensores LiDAR con la configuración deseada, y colocando un tiempo (en segundos) igual al número de muestras que se desea obtener.

Se obtendrán en el directorio donde se ejecute el programa, dos carpetas llamadas *out_16* y *out_64*. A su vez en Matlab tenemos el programa "dataser_conversion.m", el cual ordena automáticamente en carpetas como requieren las redes cycleGAN y pix2pix que vienen preparadas en el repositorio de GitHub [10].

Para preparar los datos de entrada a la red pix2pix es necesario crear un solo archivo de imagen en el que esten contenidas las dos imágenes de entrada. Para ello, el proyecto aporta un programa, el cual combina los datos y crea un dataset alineada. Se debe ejecutar el siguiente comando en la carpeta raíz, donde se haya clonado el repositorio:

```
python datasets/combine_A_and_B.py --fold_A /path/to/data/A --fold_B
/path/to/data/B --fold_AB /path/to/data
```

En concreto, para este caso y por la manera en la que los ordena el programa de Matlab, se ejecutaria en el erminal lo siguiente:

```
python datasets/combine_A_and_B.py --fold_A datasets/dataset_name/A
--fold_B datasets/dataset_name/B --fold_AB datasets/dataset_name
```

Con esto, ya se tendrían los datos de entrenamiento y test preparados para ser utilizados.

6.4 Entrenamiento y prueba de red Pix2pix

Primeramente, es necesario ejecutar visdom para visualizar el entrenamiento:

```
python -m visdom.server
```

Para entrenar la red pix2pix con la configuración que se ha entrenado para este proyecto, el comando a ejecutar es el siguiente:

```
python3 train.py --dataroot ./datasets/dataset_name --name model_name
--model pix2pix --input_nc 1 --output_nc 1 --netD pixel --preprocess none
```

Los parámetros que no se modifican en la línea de este comando son porque se utilizan los que vienen por defecto.

Luego, para testear la red una vez esta esté entrenada, se ejecuta el siguiente comando:

```
python3 test.py --dataroot ./datasets/dataset_name --name model_name
--model pix2pix --input_nc 1 --output_nc 1 --netD pixel --preprocess none
--epoch latest --netG unet_256 --norm batch
```

Para cargar una red ya entrenada, basta con copiar la carpeta del modelo generada en el directorio /checkpoints

, y tener los archivos "latest$_net_G$.pth"y"latest$_net_D$.pth".

Capítulo 7

Pliego de condiciones

A continuación se detallan de forma específica los elementos mas importantes del sistema desarrollado, centrándose en sus especificaciones técnicas, y el software utilizado en este proyecto.

7.1 Elementos físicos

- Portátil ASUS F555L Intel-Core i7-5500U
  - 3.0 GHz CPU
  - 12 GB DDR3 1333 MHz RAM
  - 1 TB HDD
  - NVIDIA GEFORCE 820M

Utilizado para desarrollo de código, depuración, estudio de resultados, creación de datasets para entrenamiento, escritura del proyecto, entre otros.

- Ordenador de Escritorio i7-8700
  - 3.2 GHz CPU
  - 32 GB DDR4 2400 MHz RAM
  - 500 GB SSD NVME
  - NVIDIA 2070 RTX

Utilizado para toma de datos en CARLA simulator, entrenamiento de las redes neuronales y para la implementación del sistema final en CARLA simulator.

7.2 Software

- Procesamiento y análisis de datos: Matlab versión 2019a, MeshLab, RVIZ
- Simulación y toma de datos: CARLA Simulator, versiones 0.9.4, 0.9.6
- Diagramas realizados con: https://app.diagrams.net/

- Procesadores de textos: Microsoft Word 2010, OverLeaf (Latex)
- Ubuntu 18.04.3 LTS (Bionic Beaver)
- Versión de ROS: Melodic
- Versión de Docker: Docker 19.03
- Entornos de desarrollo: PyCharm, Spyder, Matlab

Capítulo 8

Presupuesto

En este capítulo se incluye una estimación del coste total para la realización del proyecto, teniendo en cuenta también el vehículo real, que aunque no haya sido utilizado en el desarrollo de este proyecto, se tiene en cuenta para aplicarlo sobre este.

HARDWARE			
Concepto	Precio por Unidad	Cantidad	Subtotal
Portátil ASUS	750,00 €	1	750,00 €
Ordenador de Escritorio	2365,00 €	1	2365,00 €
Tabby EVO (vehículo)	20250,00 €	1	20250,00 €
Velodyne LiDAR Puck	7560,00 €	1	7560,00 €

Tabla 8.1: Costes de recursos Hardware

SOFTWARE			
Concepto	Precio por Unidad	Cantidad	Subtotal
Matlab	0,00 €(aportado por UAH)	1	0,00 €
CARLA	0,00 €	1	0,00 €
Microsoft Office	0,00 €(aportado por UAH	1	0,00 €
Overleaf	0,00 €	1	0,00 €
Ubuntu 18.04 LTS	0,00 €	1	0,00 €
ROS	0,00 €	1	0,00 €
PyCharm	0,00 €	1	0,00 €
Spyder	0,00 €	1	0,00 €

Tabla 8.2: Costes de recursos Software

SALARIOS			
Tarea	Precio/Hora	Horas	Coste Total
Análisis y adaptación de GANs	15,00 €/hora	50	750,00 €
Simulación y toma de datos	15,00 €/hora	30	450,00 €
Entrenamiento de GAN	15,00 €/hora	150	2250,00 €
Obtención y analisis de resultados	15,00 €/hora	50	750,00 €
Documentación y escritura	15,00 €/hora	40	600,00 €

Tabla 8.3: Costes de Personal

Costes de ejecución totales	
Concepto	Subtotal
Costes Hardware	30925,00 €
Costes Software	0,00 €
Costes Personal	4800,00 €
Subtotal final	35725,00 €

Tabla 8.4: Costes de ejecución totales

Gastos generales y beneficio industrial	
Concepto	Subtotal
Costes de ejecución material	35725,00 €
Gastos generales (13 %)	4644,25 €
Beneficio industrial (6 %)	2143,50 €
Subtotal final	6787,75 €

Tabla 8.5: Gastos generales y beneficio industrial

Persupuesto de ejecución por contrata	
Concepto	Subtotal
Costes de ejecución material	35725,00 €
Gastos generales y beneficio industrial	6787,75 €
Subtotal final	42512,75 €

Tabla 8.6: Persupuesto de ejecución por contrata

Importe final del presupuesto	
Concepto	Subtotal
Persupuesto de ejecución por contrata	35725,00 €
Porcentaje IVA	21 %
Subtotal final	8927,67 €
Importe final	51440,42 €

Tabla 8.7: Importe final del presupuesto

El presupuesto final estimado para desarrollar e implementar este Trabajo de Fin de Grado es de 51.440,42 €(cincuenta y un mil cuatrocientos cuarenta con cuarenta y dos euros).

Bibliografía

[1] "Robesafe," http://www.robesafe.com/.

[2] T. Z. A. A. E. Phillip Isola, Jun-Yan Zhu, "Image-to-image translation with conditional adversarial networks," *Publicación*, 2018.

[3] "Carla simulator," https://carla.org/.

[4] "Kitti dataset," http://www.cvlibs.net/datasets/kitti/.

[5] "Robotics operating system (ros)," https://www.ros.org/.

[6] "Docker)," https://www.docker.com/.

[7] M. M. B. X. D. W.-F. S. O. A. C. Y. B. Ian J. Goodfellow, Jean Pouget-Abadie, "Generative adversarial networks," *Publicación*, 2014.

[8] M. Z. N. E. Ahmad A. Al Sallab, Ibrahim Sobh, "Lidar sensor modeling and data augmentation with gans for autonomous driving," *Publicación*, 2019.

[9] "Velodyne hdl-64e," https://velodynelidar.com/products/hdl-64e/.

[10] J.-Y. Zhu, "Cyclegan and pix2pix in pytorch," *Repositorio*.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR